APPLE

MASCHINEN SPRACHE

DON INMAN - KURT INMAN

te·wi

APPLE

MASCHINEN-SPRACHE

Don Inman - Kurt Inman

te-wi

IMPRESSUM

Grundlage dieses Buches ist die Übersetzung der amerikanischen Originalausgabe "APPLE MACHINE LANGUAGE" by Don Inman and Kurt Inman.

LIZENZNEHMER und HERAUSGEBER:
te-wi Verlag GmbH, Theo-Prosel-Weg 1
8000 München 40

GESAMTHERSTELLUNG:
technik marketing, München
tm 3451/384
Printed in W-Germany
ISBN 3-921803-21-7

Inhaltsverzeichnis

Einleitung

Dieses Buch eröffnet APPLE-Benutzern über nur 3 BASIC-Befehle Zugang zur Maschinensprache des Mikroprozessors 6502 im APPLE-Computer.

POOKE, CALL und PEEK: mit POKE hinterlegen wir im Speicher des APPLE die Befehle eines Maschinenprogramms; mit CALL rufen wir das Maschinenprogramm vom BASIC-Programm aus auf; mit PEEK holen wir uns die Ergebnisse des abgelaufenen Maschinenprogramms ins BASIC-Programm zurück.

Vom vertrauten BASIC ausgehend, lernen wir Maschinenprogramme abzuspeichern und, Schritt für Schritt, auch die Bedeutung der Maschinenbefehle zu erkennen. Ohne Mühe erkennen wir, wie sich in den Maschinenbefehlen die Funktionsweise des Mikroprozessors widerspiegelt.

Mit Kenntnis des Zusammenhangs zwischen Maschinenbefehlen und Mikroprozessor können wir schließlich selbst Maschinenprogramme schreiben, in denen wir unmittelbar über die Funktionen des Mikroprozessors in unserem APPLE verfügen.

Dies ist der Moment, in dem wir über BASIC hinauswachsen – und mit Selbstverständlichkeit von den Möglichkeiten des APPLE SYSTEM MONITOR und APPLE MINI-ASSEMBLER für Eingabe und Edition von Maschinenprogrammen Gebrauch machen lernen.

Das Buch ist voll von Programmbeispielen für Graphik, Text, Akustik, Arithmetik und APPLE-eigenen Maschinenprogrammen, die wir mit CALL für eigene BASIC-Programme dienstbar machen.

Am Ende dieses Buches sind Ihnen vertraut:

- die Maschinensprache des Mikroprozessors 6502 in Ihrem APPLE
- der APPLE SYSTEM MONITOR
- der APPLE MINI-ASSEMBLER
- das unmittelbare Verfügen über Funktionen Ihres APPLE wie Speicherzugriffe, Bildschirmdarstellungen, Tonerzeugung usw.

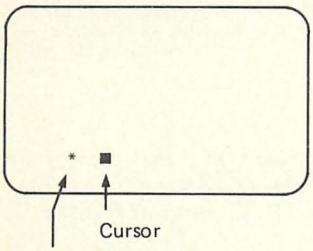
Und wofür dies alles? Ihren APPLE nur über BASIC zu kennen, ist wie Italien nur aus Postkarten zu kennen. Mit Maschinenbefehlen ist Ihnen fast jede Funktion, jeder Eingang und jeder Ausgang Ihres APPLE unmittelbar zugänglich.
Und Maschinenprogramme laufen wesentlich schneller als BASIC-Programme.

1

Applesoft II Basic - kurzgefasst

Einführung

Mit Apple-Computern kann man in mehreren Sprachen sprechen. Das Antwortsymbol auf dem Bildschirm gibt an, welche der Sprachen Ihr Apple im Augenblick versteht. Ein Sternsymbol (*) erscheint, wenn Sie mit dem Computer in Maschinensprache sprechen können. Dies ist die Muttersprache des Computers – sie muß nicht von einer Kassette oder Diskette in den Computer geladen werden. Dialoge in Maschinensprache werden von einem besonderen Programm – dem Monitor – überwacht, der in diesem Buch in Kapitel 10 besprochen wird.



zeigt an, daß der Computer Befehle in Maschinensprache erwartet

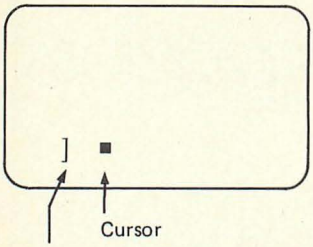
Befindet sich bei Ihnen die Sprache Applesoft Basic auf einer Einsteckkarte, dann können Sie mit Ihrem Computer auch in INTEGER BASIC sprechen. Das Antwortsymbol für INTEGER BASIC auf dem Bildschirm ist ein nach rechts weisender Pfeil (>). Die Sprache INTEGER BASIC wird nicht in diesem Buch besprochen. Informationen finden Sie im Apple II BASIC Programmierhandbuch.

Eine weitere Sprache, Applesoft II, ist eine von Apple erweiterte Version der Programmiersprache BASIC. Das Antwortsymbol auf dem Bildschirm für Applesoft II ist eine eckige Klammer (]). Für den Zugang zu Applesoft II BASIC gibt es inzwischen drei Möglichkeiten:

1. Apple II Plus-Computer mit Autostart Monitor ROM
2. Applesoft II Einsteckkarte
3. Apple Language System

In den Computern Apple II PLus ist die Sprache Applesoft II BASIC in einem ROM gespeichert (ROM steht für Read Only Memory oder Lesespeicher, dessen Inhalt nur gelesen, nicht aber von außen verändert werden kann). Arbeitet man dagegen mit Apple-Computern, die für Applesoft II eine besondere Einsteckkarte erfordern, dann hat man auch Zugang zu einem besonderen Programmsystem – dem Apple Mini-

Assembler – den wir in Kapitel 11 verwenden werden. Wir beschränken uns daher auf Computer, die Applesoft II in Form einer Einsteckkarte enthalten.



zeigt an, daß der Computer Befehle in Applesoft II BASIC erwartet

Dieses Buch ist eine Brücke für BASIC-Programmierer zur Programmierung in Maschinensprache. Wir nehmen an, daß Sie Applesoft II BASIC bereits kennengelernt haben – beginnen aber mit einer Zusammenfassung wichtiger Befehle dieser Sprache und ihrer Regeln, falls Ihre Kenntnisse Rost angesetzt hatten.

Die folgende Zusammenfassung von Applesoft II BASIC ist natürlich nicht erschöpfend, enthält aber alle Befehle und Anweisungen, die wir für dieses Buch brauchen.

Befehle

Um ein Programm aufzustellen, von Fehlern zu befreien und ablaufen zu lassen, sind bestimmte einfache Befehle notwendig. Zu den hier behandelten gehören NEW, LIST, RUN, TEXT, GRaphic, LOAD, SAVE, CONTinue, TRACE und NOTRACE.

NEW – löscht alte Programme, die sich noch im Speicher des Computers befinden. Dabei werden alle Variablen, die ein Programm benutzt hatte, gelöscht. Dieser Befehl geht der Eingabe eines neuen Programms voran.

```
] LIST
10 LET M = 50
20 PRINT M
30 LET M = M+1
40 IF M<60 THEN GOTO 20
50 END
] NEW
] LIST
```

LIST bringt ein Programm im Computer zur Darstellung

NEW löscht alle Programme im Computer

LIST – stellt das gegenwärtig gespeicherte Programm auf dem Bildschirm dar. Die folgenden Beispiele zeigen mehrere Versionen dieses Befehls. Alle Versionen gehen davon aus, daß im Computer ein Programm gespeichert ist.

Beispiele:

LIST

Darstellung des gesamten Programms im Speicher. Auf dem Bildschirm durchlaufende Darstellung bei mehr als 22 Programmzeilen.

LIST 20,100 oder LIST 20-100

Darstellung aller Programmzeilen im Bereich 20 . . . 100.

LIST -150

Darstellung aller Programmzeilen im Bereich . . . 150.
Anfang beliebige Zeilennummer < 150.

LIST 150-

Darstellung aller Programmzeilen im Bereich 150 . . . Ende.
Ende beliebige Zeilennummer > 150.

LIST 150

Programmzeile 150 darstellen.

Die fortlaufende Auflistung des Programms kann durch Drücken der Tasten CTRL und S unterbrochen werden. Erneute Tasteneingabe CTRL und S setzt die Auflistung fort. Hiermit können Sie die Programmauflistung abschnittsweise durchgeben. Mit den Tasten CTRL und C können Sie die Programmauflistung vorzeitig abbrechen.

RUN – löst den Ablauf (die Ausführung) des gegenwärtig gespeicherten Programms aus. Zunächst werden alle Variablen gelöscht. Die Ausführung beginnt mit der niedrigsten Zeilennummer des Programms, wenn nicht im Befehl RUN eine abweichende Zeilennummer angegeben wurde (Beispiel 2).

Beispiele:

RUN

Das Programm mit der niedrigsten Anfangszeilennummer wird gestartet.

RUN 200

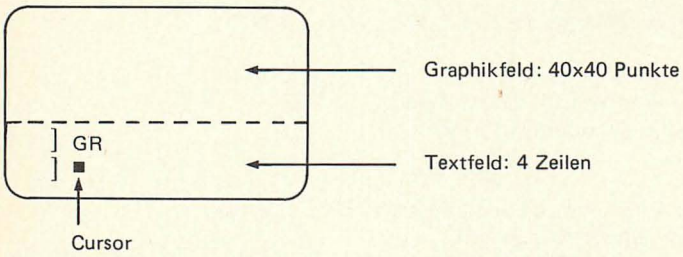
Das Programm mit der Anfangszeilennummer 200 wird gestartet. (Es können also mehrere BASIC-Programme gleichzeitig im Speicher stehen).

TEXT – legt das Bildschirmformat für Textdarstellungen auf maximal 40 Zeichen je Zeile und 24 Zeilen fest. Dies ist das normale Bildschirmformat für Arbeiten in Applesoft II BASIC. Dieser Befehl dient der Rückkehr zu Textdarstellungen nach vorausgegangenen graphischen Darstellungen. Er kann auch als Programmanweisung, d.h. innerhalb eines Programms, ausgeführt werden.

GR – legt das Bildschirmformat für Graphikdarstellungen auf ein Gitternetz von 40 x 40 Punkten fest. Der Bildschirm wird gelöscht, der Bildschirmhintergrund ist dunkel, und der Cursor wird an den Anfang eines 4zeiligen Textfensters unten am Bildschirm bewegt. Die Farbe der Graphikdarstellungen wird automatisch auf schwarz (Color = 0) gesetzt. Graphikdarstellungen Schwarz-auf-Schwarz müssen

durch Eingabe eines Zahlenwerts für COLOR in COLOR-auf-Schwarz geändert werden.

Beispiel:
GR und 'RETURN' eingeben



LOAD – lädt ein Apple-Programm von einer Magnetbandkassette oder Diskette in den Speicher des Computers. Bei Cassettengeräten muß der Benutzer das Magnetband bis zum Anfang des gewünschten Programms spulen und die Taste PLAY gedrückt haben, ehe der Befehl LOAD eingegeben wird. Der Computer meldet sich mit einem Signalton, sobald das Programm gefunden ist. Ein zweiter Signalton zeigt an, daß das Programm erfolgreich in den Computer geladen wurde. Zugleich erscheint das Antwortsymbol (]) von Applesoft II auf dem Bildschirm. Ein LOAD-Befehl kann nur mit der Taste RESET unterbrochen werden – oder durch Auschalten des Computers.

```
] LOAD  ← Ihre Eingabe
] ■     ← Apple's Antwort nach erfolgreichem Laden des Programms
```

SAVE – speichert ein gegenwärtig im Computer stehendes Programm auf einer Magnetbandkassette oder Diskette ab. Bei Kassettengeräten muß der Benutzer zuvor die Tasten RECORD und PLAY gedrückt haben, ehe der Befehl SAVE eingegeben wird. Signaltöne des Computers zeigen Anfang und Ende der Abspeicherung an.

CONT – setzt einen Programmablauf fort, der durch STOP, END oder CTRL C unterbrochen worden war. Der Befehl verändert keine Speicherinhalte. CONT kann nicht eingegeben werden, wenn Sie (1) Programmzeilen verändert, hinzugefügt oder gelöscht haben (2) eine Fehlermeldung auf dem Bildschirm empfangen haben, die auch den Programmablauf unterbrach.

TRACE – stellt die Zeilennummern des Programms in der Reihenfolge ihrer Ausführung auf dem Bildschirm dar. Mit diesem Befehl arbeitet man bei der Fehlersuche. Aus der Reihenfolge der Zeilennummern läßt sich ersehen, ob das Programm Anweisungen in der gewünschten Folge ausführt. TRACE-Darstellungen werden durch den Befehl NOTRACE wieder ausgeschaltet.

Beispiel:

TRACE und 'RETURN' eingeben.
Der Computer befindet sich jetzt im Darstellungsmodus "TRACE".

RUN
Die Zeilenweise Programmausführung wird unter Angabe von Zeilennummern auf dem Bildschirm dargestellt.

NOTRACE – hebt den oben besprochenen Befehl TRACE wieder auf.

Beispiel:

NOTRACE und 'RETURN' eingeben.
Der Darstellungsmodus "TRACE" ist wieder gelöscht.

Zuweisungen

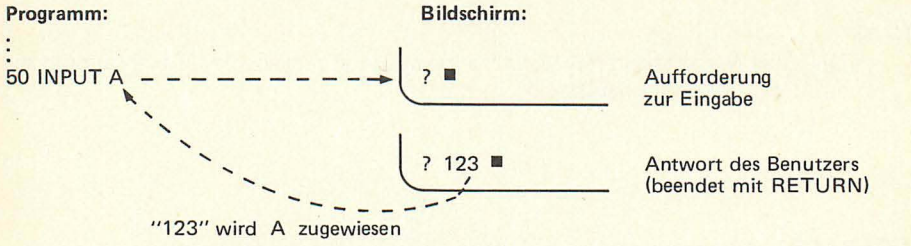
Es gibt verschiedene Möglichkeiten, Daten (Zahlen und Text) Variablen zuzuweisen. In diesem Abschnitt werden wir Zuweisungen mit LET, INPUT, READ und GET besprechen. Ebenso werden die Anweisungen DATA . . . RESTORE beschrieben, die in Verbindung mit der Anweisung READ verwendet werden.

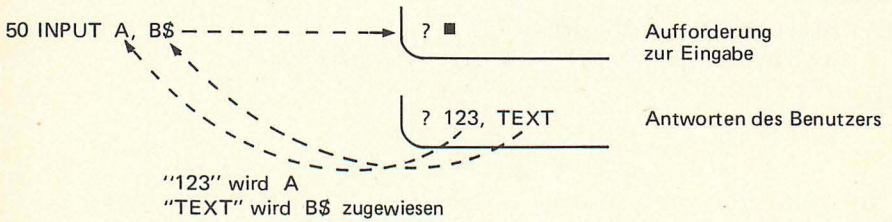
LET – kann bei der Zuweisung von Werten zu Variablen verwendet werden. Eine Zuweisung ist auch ohne Verwendung des Worts LET möglich, wie Programmzeilen 50 und 60 der folgenden Beispiele zeigen.

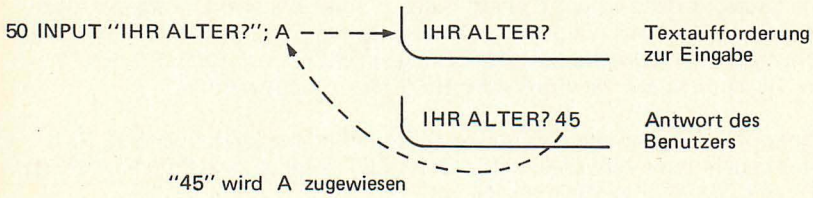
Beispiele:

```
10 LET M = 50          ← Zahlenwert 50 der Variablen M zuweisen
20 FOR X = 1 TO 9
30 LET A$ = "APPLES"   ← Text "APPLES" der Variablen A$ zuweisen.
40 LET M = M+1         ← Variable M auf den Wert M+1 erhöhen
50 B = 1
60 B$ = "PER CARTON"
70 PRINT M;A$,B;B$     ←
80 NEXT X              ← 'LET' kann entfallen
```

INPUT – weist einer Variablen während des Programmablaufs Werte zu. Bei dieser Anweisung wird der Programmablauf unterbrochen und der Benutzer über den Bildschirm aufgefordert, durch Eingabe von Zahlen oder Text den Variablen in INPUT Werte zuzuweisen.







READ – weist den Computer an, aus einer DATA-Liste einen Wert zu lesen und der Variablen in READ zuzuweisen. (Bei erstmaliger Ausführung von READ wird der erste Wert in der DATA-Liste zur Zuweisung verwendet. Bei der zweiten Ausführung, der zweite Wert usw.). Siehe DATA für ein Anwendungsbeispiel.

DATA – ermöglicht das Deponieren von Daten im Programm. Ein Programm kann mehr als eine DATA-Anweisung enthalten. Die in einer oder mehreren DATA-Listen aufgeführten Werte werden der Reihe nach von READ für Zuweisungen gelesen: Erst die Werte der ersten DATA-Anweisung, dann die Werte einer folgenden DATA-Anweisung.

Beispiel:

```
110 FOR X = 1 TO 10
120  READY                    ← 'READ' liest der Reihe nach die Zahlen in 'DATA':
130 NEXT X                        10
140 DATA 10,30,20,40,50           30
150 DATA 60,80,90,70,100          20
                                  40
                                  50
                                  60
                                  80
                                  90
                                  70
                                  100
```

RESTORE – setzt das Lesen von Werten aus DATA-Listen auf den ersten Wert in der ersten DATA-Anweisung zurück.

Beispiel:

```
100 FOR X = 1 TO 5
110  READ Y: PRINT Y      ← Liest und druckt: 10,30,50,20,40
120 NEXT X
130 RESTORE               ← Setzt Lesezeiger auf Anfang der 'DATA'-Liste zurück
140 FOR Z = 1 TO 10
150  READ W: PRINT W      ← Liest und druckt: 10,30,50,20,40,60,80,100,70,90
160 NEXT Z
170 DATA 10,30,50,20,40
180 DATA 60,80,100,70,90
```

GET – liest ein einzelnes Zeichen, das am Tastenfeld eingegeben wurde. Der Programmablauf wird wie bei INPUT unterbrochen und die Eingabe abgewartet. GET stellt keine Aufforderung zur Eingabe am Bildschirm dar. Das eingegebene Zeichen wird nicht auf dem Bildschirm dargestellt und muß nicht mit RETURN abgeschlossen werden.

Beispiel:

```
200 GET H$                        ← 'GET' weist Tasteneingabe der Variablen H$ zu
210 IF H$ = "Y" THEN GOTO 500     ← Abfrage: "War Tasteneingabe ein Y?"
220 GOTO 100
```

Bildschirm-Darstellungen

Zur Gestaltung von Bildschirm-Darstellungen stehen uns die Befehle PRINT, INVERSE, FLASH, NORMAL, HOME und SPC zur Verfügung.

PRINT – dient zur Darstellung und Gestaltung von Text- und Zahlenwiedergaben auf dem Bildschirm. Die folgenden Beispiele zeigen Anwendungsmöglichkeiten von PRINT.

Beispiel:

```
320 PRINT                    ← bewirkt Einfügen einer Leerzeile auf dem Bildschirm
```

```
300 A = 5: B = 6
310 PRINT A                  ← Wert von A darstellen
320 PRINT                    ← Leerzeile einfügen
330 PRINT B                  ← Wert von B darstellen

    Bildschirm:
    5                        ← Wert von A
                             ← Leerzeile durch 'PRINT' in Zeile 320
    6                        ← Wert von B
```

```
400 PRINT "EIN TEXT"         ← Inhalt von ". . ." darstellen

    Bildschirm:
    EIN TEXT
```

```
410 A = 5: B = 6
420 PRINT A,B                ← Komma "," als Trennzeichen

    Bildschirm:
    5          6             ← Wirkung von "," in PRINT A,B
```

```
410 A = 5: B = 6
420 PRINT A;B                ← Semikolon ";" als Trennzeichen

    Bildschirm:
    56                       ← Wirkung von ";" in PRINT A;B
```

```
410 A$ = "A" : A = 5
420 PRINT A$; A

    Bildschirm:
    A = 5                    ← A$ führt zur Darstellung A = .
                               A führt zur Darstellung 5.
```

```
410 A$ = "A=" : A = 5
420 B$ = "B=" : B = 6
430 PRINT A$; A,             ← Komma "," am Ende des 'PRINT'-Befehls
440 PRINT B$; B

    Bildschirm:
    A = 5      B = 6         ← durch Abschluß eines 'PRINT'-Befehls mit "," wird der
                               folgende 'PRINT'-Befehl auf derselben Zeile dargestellt
```

```
410 A$ = "A=" : A = 5
420 PRINT TAB (10) A$; A     ← mit TAB = Tabulator wird der Cursor verschoben – hier
                               auf Spalte 10 des 40spaltigen Bildschirms

    Bildschirm:
                       A = 5
    ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
    1                  10
```

INVERSE – bewirkt eine Schwarz-Weiß-Vertauschung der Bildschirmdarstellung: Es werden schwarze Zeichen auf weißem Hintergrund dargestellt.

Programm: **Bildschirm:**

100 PRINT "APPLE" - - - - - →

100 INVERSE
110 PRINT "APPLE" - - - - - →

FLASH – läßt Bildschirmdarstellungen zwischen Weiß-auf-Schwarz und Schwarz-auf-Weiß schwanken. Hiermit kann beispielsweise die Aufmerksamkeit des Benutzers auf eine Nachricht des Programms gelenkt werden.

Programm: **Bildschirm:**

70 FLASH
80 PRINT "JETZT BLINKT'S"

pulsierender Wechsel zwischen
normaler und inverser Darstellung

NORMAL – hebt die Anweisung FLASH auf. Bildschirmdarstellungen erfolgen wieder als weiße Zeichen auf schwarzem Hintergrund ohne Wechsel in der Darstellungsart.

Beispiel:

90 NORMAL ← Rückkehr zur normalen Darstellungsform

HOME – löscht den gesamten Bildschirm und bringt den CURSOR in Ausgangsstellung (Bildschirmrand links oben). HOME kann in Graphik- und Textmode verwendet werden.

SPC (X) – wird innerhalb von PRINT-Anweisungen verwendet und ermöglicht, den Spaltenabstand aufeinanderfolgender PRINT-Darstellungen festzulegen. X bezeichnet dezimal die Anzahl der Zwischenräume. Die Anweisung wird in der Form ; SPC (X); in PRINT-Anweisungen verwendet.

Programm:

400 A = 1111 : B = 2222
410 PRINT A; SPC(2); B ← Mit SPC = space cursor wird der Cursor von seiner augenblicklichen Lage um die angegebene Zahl von Spalten nach rechts verschoben

Bildschirm:

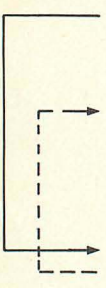
1 1 1 1 □□ 2 2 2 2
↑↑
SPC(2) ← SPC(2) bewegt den Cursor □ um 2 Spalten weiter, ehe die nächste Darstellung beginnt

Schleifen, Verzweigungen und Unterprogramme

In diesem Abschnitt fassen wir BASIC-Anweisungen zusammen, mit denen sich Programmabschnitte wiederholt durchlaufen (FOR . . . NEXT), Sprünge ausführen (GOTO, ON . . . GOTO, IF . . . THEN) und Unterprogramme aufrufen lassen (GOSUB . . . RETURN).

GOTO – bewirkt einen Sprung von der Programmzeile, auf der GOTO steht, zu der nach GOTO angegebenen Programmzeile.

Beispiel:



```
30 GOTO 200          ← Programmfortsetzung mit Zeile 200
.                      alle dazwischen liegende Zeilen werden
.                      übersprungen
.
70 A = 73
80 PRINT A
90 END
.
.
.
200 PRINT "THE END"
210 GOTO 70          ← Rücksprung zu Programmzeile 70
```

ON . . . GOTO – wertet einen nach ON folgenden arithmetischen Ausdruck aus und springt mit GOTO abhängig vom Ergebnis des arithmetischen Ausdrucks zu einer von drei Programmzeilen, die nach GOTO aufgeführt sind.

Beispiel:

```
.
.
.
150 ON INT(B/100) GOTO 200,300,400
160                  ← Programmfortsetzung abhängig vom ganz-
.                      zahligen Ergebnis 0, 1, 2, 3 der Auswertung
.                      eines Ausdrucks nach ON, hier des Aus-
.                      drucks INT(B/100):
                       0 : Fortsetzung mit nächster Zeile (Zeile 160)
                       1 : Sprung zu Zeile 200
                       2 : Sprung zu Zeile 300
                       3 : Sprung zu Zeile 400
```

IF . . . THEN – prüft eine nach IF aufgeführte Bedingung. Ist die Bedingung erfüllt, dann wird die Anweisung nach THEN ausgeführt. Handelt es sich hierbei um die Anweisung GOTO, dann spricht man von einem bedingten Programmsprung. Ist die Bedingung nach IF nicht erfüllt, setzt das Programm mit der nächsten Programmzeile fort, ohne die Anweisung nach THEN auszuführen.

Beispiele:

```
200 IF X>5 THEN 400              ← Programmfortsetzung mit Zeile 400,
                                   WENN X größer als 5

300 IF X>5 AND Y<10 THEN 400     ← Verzweigung nach Zeile 400, WENN X
                                   größer als 5 UND Y kleiner als 10
```

FOR . . . NEXT – dient der Bildung von Programmschleifen zum wiederholten Durchlaufen eines zwischen FOR und NEXT stehenden Programmabschnitts.

Programm:

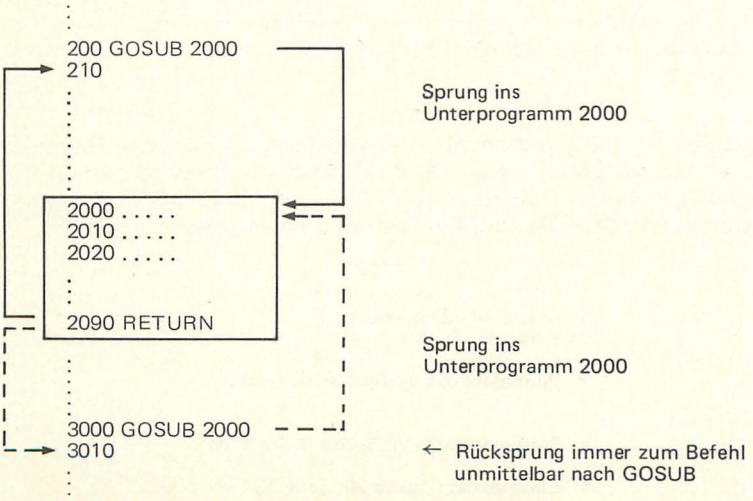
```
20 FOR X = 1 TO 5          → läßt X die Werte 1, 2, 3, 4, 5 durchlaufen
30 PRINT X; 10             → liefert zu dem X-Wert eine Darstellung
40 NEXT X                  → erhöht X vor Rückkehr zu FOR auf den Wert X+1

20 FOR N = -10 TO -6 STEP 2  → läßt X die Werte -10, -8, -6, -4, -2 in 2er-Schritten
30 PRINT N                     (STEP 2) durchlaufen
40 NEXT N
```

Bildschirm:

```
 110          ← Darstellungen von PRINT X; 10 für X = 1, 2, 3, 4, 5
 210
 310
 410
 510

-10           ← Darstellungen von PRINT N für N = -10, -8, -6, -4, -2
 -8
 -6
 -4
 -2
```

GOSUB . . . RETURN – bewirkt den Sprung in ein Unterprogramm, dessen Anfangszeile nach GOSUB aufgeführt ist. Mit RETURN am Ende des Unterprogramms erfolgt die Rückkehr zur nächsten, nach GOSUB folgenden Programmzeile des Hauptprogramms. GOSUB steht im Hauptprogramm, RETURN steht am Ende des Unterprogramms.

Beispiel:



Graphikdarstellungen

Anweisungen zur Darstellung graphischer Symbole auf dem Bildschirm sind GR, COLOR, PLOT, HLIN, VLIN, PDL und TEXT. Graphikdarstellungen auf dem Bildschirm unterscheiden sich erheblich von Textdarstellungen. Wir zeigen, wie man zwischen beiden Betriebsarten wechseln kann.

GR – bewirkt den Übergang von hochauflösenden Textdarstellungen zu geringauflösenden Graphikdarstellungen auf dem Bildschirm.

COLOR – bestimmt die Farbe von Bildschirmdarstellungen im Graphikmode. Die Anweisung GR setzt automatisch COLOR = 0 = Schwarz. Den anderen Farben sind folgende Werte zugewiesen:

Farbcode		Farbe
dez	hex	
0	0	Schwarz
1	1	Magenta
2	2	Dunkelblau
3	3	Hellpurpur
4	4	Dunkelgrün
5	5	Grau
6	6	Mittelblau
7	7	Hellblau
8	8	Braun
9	9	Orange
10	A	Grau
11	B	Pink
12	C	Grün
13	D	Gelb
14	E	Blau/Grün
15	F	Weiß

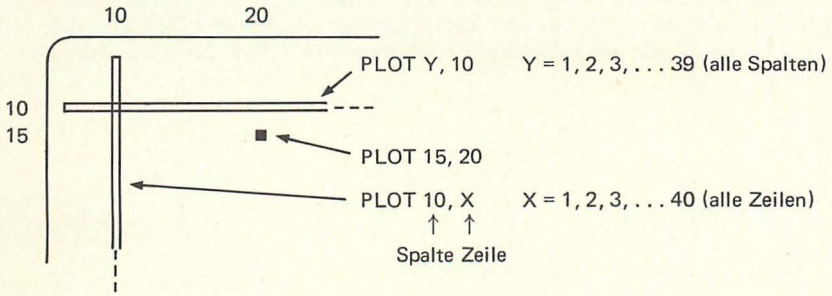
Farbcodes für den niedrigauflösenden Graphikmode.

PLOT – spricht einen der 1600 Rasterpunkte des 40zeiligen x 40spaltigen Graphikbildschirms an. Die Koordinaten (Spalte, Zeile) des Rasterpunkts werden nach PLOT angegeben. Die Farbe des Rasterpunkts wird zuvor durch COLOR festgelegt. Die drei Anweisungen GR, COLOR, PLOT werden zusammen gegeben.

Programm:

```
10 GR                    ← Graphikmode einstellen
20 COLOR = 15            ← Farbe der Punkte
30 FOR X = 1 TO 40
40 PLOT 10, X            ← Punktdarstellung: Spalte 10, Zeile X
50 NEXT X
60 FOR Y = 1 TO 39
70 PLOT Y, 10            ← Punktedarstellung: Spalte Y, Zeile 10
80 NEXT Y
90 PLOT 20, 15           ← Einzelpunkt: Spalte 20, Zeile 15
```

Bildschirm:

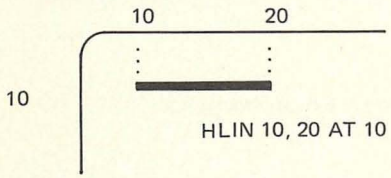


HLIN – zeichnet eine horizontale Linie. Anfangs- und Endspalte der horizontalen Linie sowie die Zeile, in der sie liegt, werden nach HLIN angegeben.

Programm:

```
10 GR
20 COLOR = 15
30 HLIN 10, 20 AT 10
       ↑  ↑     ↑
   Anfang Ende Zeile
```

Bildschirm:

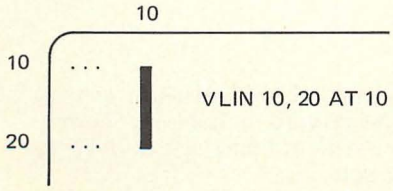


VLIN – zeichnet eine vertikale Linie. Anfangs- und Endzeile dieser Linie sowie die Spalte, in der sie liegt, werden nach VLIN angegeben.

Programm:

```
10 GR
20 COLOR = 15
30 VLIN 10, 20 AT 10
       ↑  ↑     ↑
   Anfang Ende Spalte
```

Bildschirm:

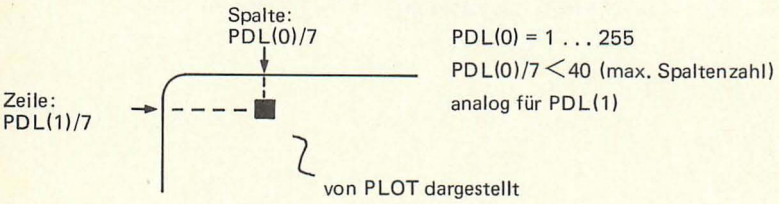


PDL – liest die augenblickliche Einstellung extern angeschlossener Spielkontrollen. Die Einstellungen werden durch die Werte 0 bis 255 dargestellt. Mit PDL (0) und PDL (1) kann jede der beiden Spielkontrollen abgefragt werden. Die Spielkontrollen eignen sich zur manuell verstellbaren Darstellung von Punkten im Graphikmode des Bildschirms.

Programm:

```
10 GR
20 COLOR = 15
30 PLOT  PDL(0)/7, PDL (1) 7
:              ↑        ↑
:           Spalte     Zeile
```

Bildschirm:

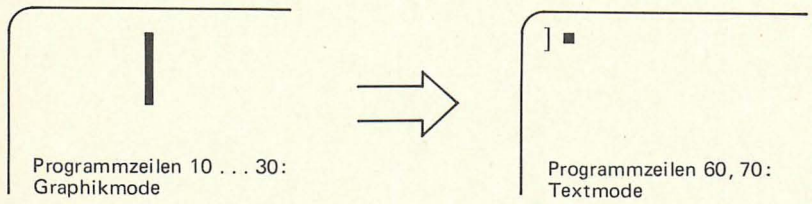


TEXT – bewirkt die Rückkehr von Graphikdarstellungen zu Textdarstellungen auf dem Bildschirm.

Programm:

```
10 GR                    ← Übergang TEXTmode → GRaphikmode
20 COLOR = 14
30 VLIN 6, 14 AT 12
40 FOR X = 1 TO 3000
50 NEXT X                ← (Zeitverzögerung)
60 TEXT                  ← Rückkehr GRaphikmode → TEXTmode
70 HOME
```

Bildschirm:



Vergleichende und logische Ausdrücke

Zwei Zahlenwerte können durch eine der folgenden BASIC-Anweisungen miteinander verglichen werden. Das Ergebnis des Vergleichs wird im Computer durch eine logische 1 (das angenommene Größenverhältnis trifft zu) oder eine 0 (das angenommene Größenverhältnis trifft nicht zu) gespeichert.

Mathematische Relationen

Symbole	Bedeutung	Mögliche Antworten
A = B	A GLEICH B ?	1: WAHR die Relation trifft zu
A > B	A GRÖSSER ALS B ?	
A < B	A KLEINER B ?	
A >= B	A GRÖSSER ODER GLEICH B ?	0: NICHT WAHR die Relation trifft nicht zu
A <= B	A KLEINER ODER GLEICH B ?	
A # B	A UNGLEICH B ?	

Die Ergebnisse 0 oder 1 mehrerer Vergleiche können anschließend mit einer der folgenden Anweisungen logisch verknüpft werden.

Logische Relationen

A AND B = WAHR, wenn A UND B wahr sind
A OR B = WAHR, wenn A ODER B wahr ist
NOT B = WAHR, wenn B NICHT WAHR ist

Vergleichende und logische Ausdrücke können in IF . . . THEN-Anweisungen dazu dienen, einen Programmablauf abhängig von den Größenverhältnissen von Zahlenwerten zu steuern.

Beispiele:

```
200   IF A #B THEN GOTO 2000
300   IF A<=B AND C<=D THEN 2000
400   B = 0
500   IF NOT B THEN 2000
```

Der Sprung zu Programmzeile 2000 erfolgt, wenn

1. A UNGLEICH B ist
2. A KLEINER ODER GLEICH B UND gleichzeitig C KLEINER ODER GLEICH D ist
3. B NICHT WAHR ist, d.h. den Wert 0 hat

Rangfolge arithmetischer und logischer Operationen

Der Computer wertet mathematische und logische Operationen in Ausdrücken nach einer bestimmten Rangfolge aus. Die Rangfolge ist in der folgenden Liste zusammengestellt: Erfordert ein Ausdruck mehrere Operationen, dann wird die in der Liste zuerst genannte Operation zuerst ausgeführt usw..

Rang ↓	Operation ↓	Erklärung
1.	( )	Klammerausdruck auswerten
2.	NOT	Negieren
3.	^	Potenzieren (z.B. 2^5)
4.	*, /	Multiplizieren oder Dividieren
5.	+, –	Addieren oder Subtrahieren
6.	=, >, <, >=, <=, #	Vergleichen
7.	AND	Logische UND-Verknüpfung
8.	OR	Logische ODER-Verknüpfung

Funktionen für Zahlen und Text

BASIC enthält zahlreiche Anweisungen zur Manipulation von Zahlen und Text. In diesem Buch werden wir nur die Funktionen ASC, CHR#, LEFT# und INT benutzen.

ASC – meldet den ASCII-Code des Anfangsbuchstabens eines nach ASC in Klammern folgenden Textes als Dezimalzahl zurück. Hier die ASCII-Codes:

Die 128 ASCII-Codes

dez	hex	Zeichen	dez	hex	Zeichen	dez	hex	Zeichen	dez	hex	Zeichen
0	00	NUL	32	20	SP	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	–	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Die Anwendung von ASC zeigen die folgenden Beispiele:

```
100 PRINT ASC ("YES")                    ← stellt die Zahl 89 dar
                                           (Y hat den ASCII-Code 89)

200 GET H$                               ← Tasteneingabe lesen
210 FOR I = 65 TO 90
220 IF ASC (H$) = I THEN 250,            prüfen, ob Tasteneingabe einen der
230 NEXT I,                              Buchstaben A . . . Z darstellt
240 PRINT "IHRE EINGABE IST KEIN BUCHSTABE"
250
:
```

**CHR$** – liefert das ASCII-Zeichen zu einer nach CHR$ in Klammern angegebenen Dezimalzahl. Der Wert dieser Dezimalzahl muß zwischen 0 und 255 liegen.

Beispiel:

```
300 H = 14
310 PRINT CHR$ (H+55)        ← stellt das Zeichen E (ASCII-Code 69) dar
                               H = 14. 14+55 = 69. CHR$ (69) → E
```

**LEFT$** – liefert die linksstehenden Zeichen eines nach LEFT$ in Klammern folgenden Texts zurück. Die Anzahl der gewünschten von links gezählten Zeichen wird als Dezimalzahl ebenfalls in der Klammer angegeben.

Beispiele:

```
200 PRINT LEFT$ ("ABCDEFGH", 3)      ← stellt ABC dar
                                       (die 3 ersten Zeichen von links)

300 INPUT H$                         ← liest eine Tasteneingabe
310 IF LEFT$ (H$,1) = "J" THEN 2000  ← isoliert den 1. Buchstaben
                                       wenn 1. Buchstabe ein "J", dann
                                       Verzweigung nach Zeile 2000
```

INT – liefert den größten, ganzzahligen Wert einer nach INT in Klammern folgenden Zahl.

Beispiele:

```
100 X = INT (A/3)
      If A = 5, A/3 = 1.66667 and INT (A/3) = 1
      If A = 1, A/3 = 0.333333 and INT (A/3) = 0
      If A = 15, A/3 = 5 and INT (A/3) = 5
      If A = –5, A/3 = –1.66667 and INT (A/3) = –2
```

Besondere BASIC-Anweisungen

Für den Übergang von der Programmiersprache BASIC zur Maschinensprache werden wir ständig die BASIC-Befehle POKE, PEEK und CALL benutzen. Während der Computer bei BASIC-Programm die Reihenfolge der Befehlsausführungen an den aufsteigenden Zeilennummern erkennt, folgt er bei Maschinenprogrammen le-

diglich den aufeinanderfolgenden Speichereintragungen. Die Programmausführung beginnt an einer angegebenen Speicheradresse. Die Befehle werden gewöhnlich in der Reihenfolge ausgeführt, in der sie im Speicher stehen. POKE gestattet, eine Folge von Maschinenbefehlen im Speicher abzulegen.

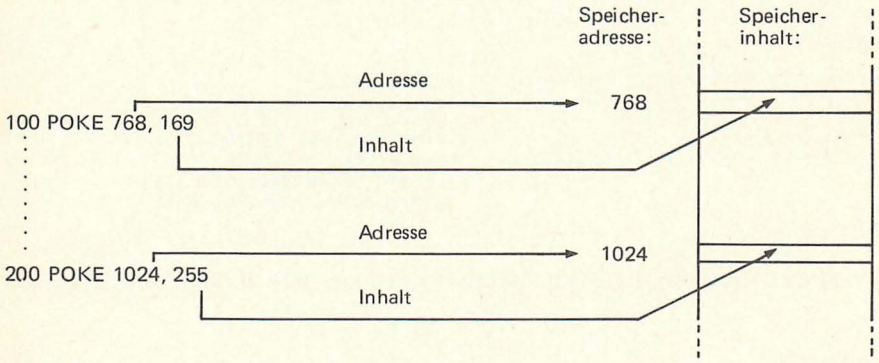
POKE – lädt Befehle und Daten eines Maschinenprogramms in den vorgesehenen Speicherbereich. POKE wird als BASIC-Anweisung in unserem Betriebssystem BBS (siehe Kapitel 2) benutzt. Befehle und Daten des Maschinenprogramms werden in POKE-Befehlen als Dezimalzahlen angegeben.

```
POKE adresse, daten
       ↑      ↑
     dezimal
     angegeben
```

adresse: eine der Speicheradressen 0 . . . 65535
daten: eine der Zahlen 0 . . . 255

Befehle und Daten eines Maschinenprogramms werden der Reihe nach mit POKE-Befehlen in den vorgesehenen Speicherbereich geladen.

Beispiel:



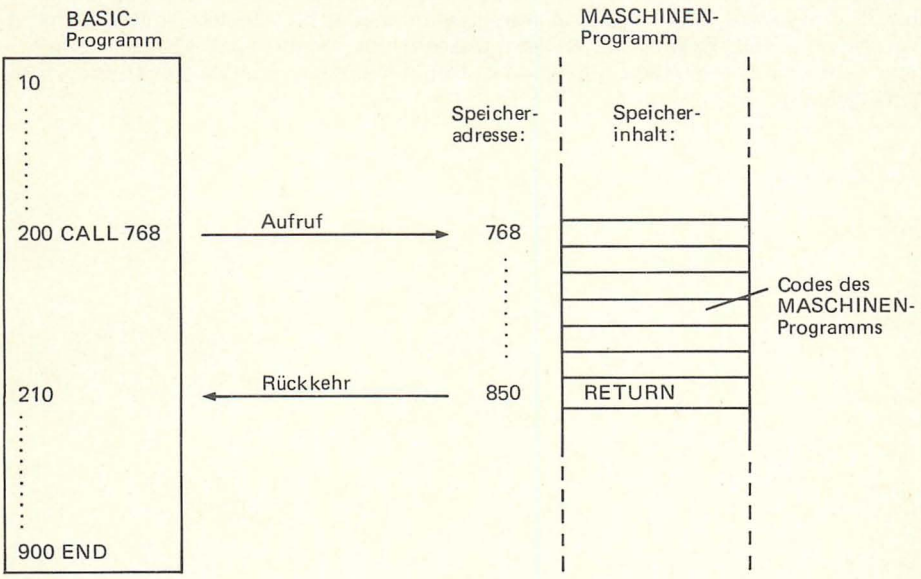
CALL – ruft von einem BASIC-Programm aus ein Maschinenprogramm auf und läßt es ablaufen.

```
CALL adresse
       ↑
     dezimal
     angeben
```

adresse: Anfangsadresse des Maschinenprogramms,
bevorzugt eine der Speicheradressen 768 . . . 1023

Die Anfangsadresse des aufgerufenen Maschinenprogramms wird nach CALL als Dezimalzahl angegeben.

Beispiel:

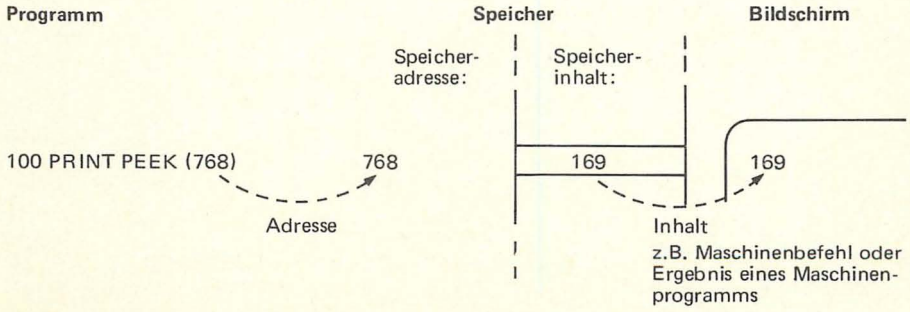


PEEK – liest den Inhalt einer Speicherzelle, deren Adresse nach PEEK in Klammern als Dezimalzahl angegeben wurde.

PEEK (adresse)
↑
dezimal
angeben

adresse: eine der Speicheradressen 0 . . . 65 535

In Verbindung mit einer PRINT-Anweisung kann PEEK einen Speicherinhalt auf dem Bildschirm darstellen.
Mit PEEK-Befehlen kann daher ein abgespeichertes Maschinenprogramm zur Überprüfung auf dem Bildschirm dargestellt werden. Ebenso können die im Speicher stehenden Ergebnisse eines Maschinenprogramms mit PEEK gelesen werden.



Die Befehle POKE, CALL und PEEK werden also wie folgt für Aufbau, Ablauf und Ergebnislesen eines Maschinenprogramms benutzt: Mit POKE wird von einem BASIC-Programm aus ein Maschinenprogramm im Speicher abgelegt, mit CALL wird von einem BASIC-Programm aus dieses Maschinenprogramm zum Ablauf gebracht und mit PEEK werden die Ergebnisse des Maschinenprogramms ins BASIC-Programm zurückgeholt.

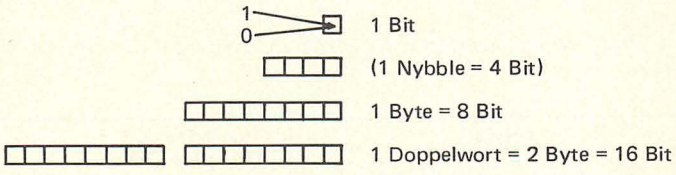
2
Alles über Zeichen

Einleitung

Beim Übergang von BASIC zu MASCHINENSPRACHE begegnen wir neuen Formen der Zeichen- und Zahlendarstellung, die uns das Arbeiten mit dem Mikroprozessor im APPLE erleichtern.

Während in BASIC Dezimalzahlen und ASCII-Codes genügen, werden wir in Maschinensprache der größeren Nähe zum Mikroprozessor dadurch gerecht, daß wir zusätzlich mit hexadezimalen und binären Schreibweisen arbeiten.

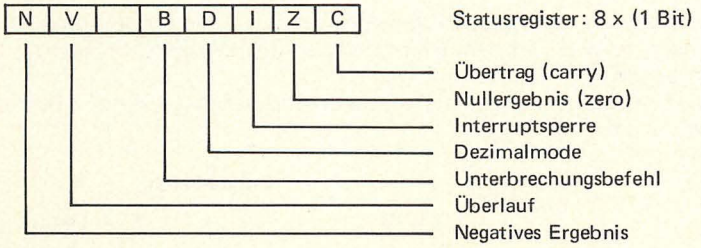
Bits, Bytes, Doppelworte



Bit

Das Bit stellt die beiden einzigen Zustände dar, mit denen der Mikroprozessor arbeitet: 0 oder 1.

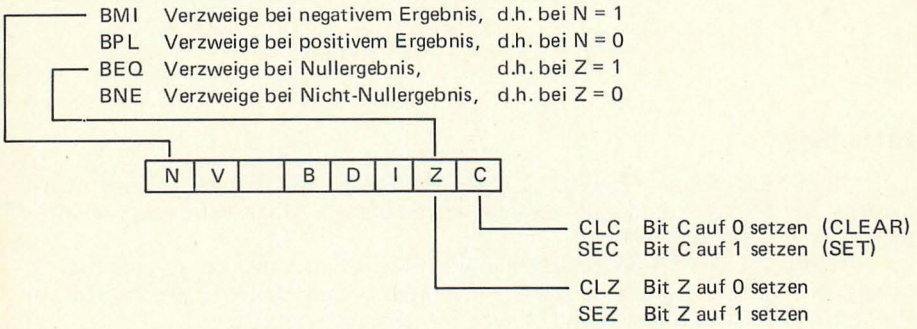
Beispielsweise verfügt der Mikroprozessor 6502 im APPLE-Computer über ein Statusregister, das aus 8 einzelnen Bits besteht. In jedem dieser Bits kann der Mikroprozessor durch 0 oder 1 anzeigen, wie das Ergebnis seiner Operationen ausgefallen ist:



Hatte eine Subtraktion ein negatives Ergebnis, dann wird Bit 7 (N) auf den

Wert 1 gesetzt. Führte die gleiche Subtraktion zum Ergebnis 0, dann wird Bit 1 (Z) auf den Wert 1 gesetzt.

Verzweigungsbefehle in Maschinenprogrammen machen ausgiebig von diesen 1-Bit-Informationen des Mikroprozessors Gebrauch. Hier als Beispiel Maschinenbefehle, die die einzelnen Bits des Statusworts benutzen bzw. verändern:



Byte oder 8-Bit-Wort

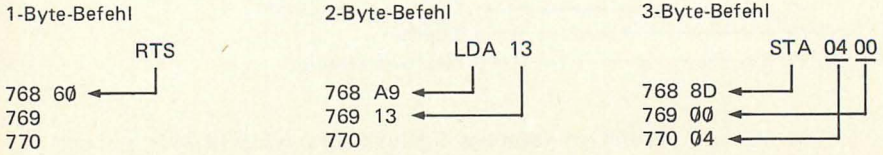
Der APPLE-Computer arbeitet mit einem 8-Bit-Mikroprozessor. Dieser sendet und empfängt Daten in Form von 8-Bit-Worten. Ein 8-Bit-Wort wird auch 1 Byte genannt.

Mit 1 Byte lassen sich $2^8 = 256$ Zeichen oder die Dezimalzahlen 0 . . . 255 darstellen:

1 Byte								darstellbare Dezimalzahlen	
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	1	1	$0 \times 2^7 + \ldots + 1.2^0 = 1$
0	0	0	0	0	0	1	0	2	
				.				.	
				.				.	
				.				.	
1	0	0	0	0	0	0	0	128	$1 \times 2^7 + 0 \times 2^6 + \ldots = 128$
				.				.	
				.				.	
				.				.	
1	1	1	1	1	1	1	1	255	$1 \times 2^7 + 1 \times 2^6 + \ldots = 255$

Abgesehen von Angaben zum Speichervermögen unseres APPLE-Computers, z.B. 64K-Byte Speicher, begegnen wir dem Begriff "Byte" beim Arbeiten in Maschinensprache ununterbrochen.

Beispielsweise gibt es Maschinenbefehle, für deren Speicherung 1 Byte, 2 Bytes oder 3 Bytes erforderlich sind:



(768, 769, 770 stellen Speicheradressen dar, in denen die Codes 60 bzw. A9 13 bzw. 8D 00 04 von Maschinenbefehlen abgespeichert sind).

Doppelwort oder 16-Bit-Wort

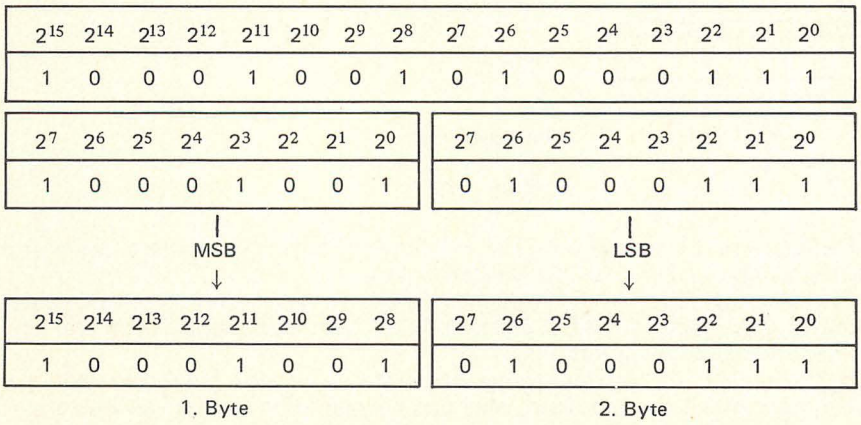
In Maschinenprogrammen manipulieren wir zwar Daten in 8-Bit-Blöcken, genannt Byte – aber damit sind wir nicht auf die Dezimalzahlen 0 . . . 255 beschränkt. Mit zwei aufeinanderfolgenden Speicherplätzen können wir ein 16-Bit-Wort bilden, genannt Doppelwort.

Mit einem Doppelwort lassen sich 2^{16} = 65536 Zeichen oder die Zahlen 0 . . . 65535 darstellen.

1 Doppelwort		darstellbare Dezimalzahlen	
$2^{15}2^{14}2^{13}-2^{8}$	$2^{7}-2^{3}2^{2}2^{1}2^{0}$		
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0	
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1	1	$0 \times 2^{15} + \ldots + 1 \times 2^{0} = 1$
0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 1	2	
	.	.	
	.	.	
	.	.	
1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	32 768	$1 \times 2^{15} + 0 \times 2^{14} + \ldots = 32\,768$
	.	.	
	.	.	
	.	.	
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	65 535	$1 \times 2^{15} + 1 \times 2^{14} + \ldots + 1 \times 2^{0} = 65\,535$

Zur Bezeichnung des Stellenwerts zweier als Doppelwort interpretierter Bytes findet man oft die Bezeichnungen MSB, LSB. Die Binärstellen des MSB (most significant byte, höchstwertiges Byte) haben das Gewicht 2^{15} . . . 2^{8}. Die Binärstellen des LSB (least significant byte, niedrigstwertiges Byte) behalten ihre Gewichte 2^{7} . . . 2^{0}.

16-Bit-Zahl, realisiert als Doppelwort:



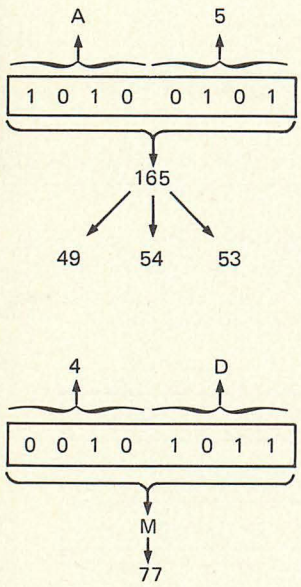
In Maschinenprogrammen arbeiten wir sehr häufig mit Doppelworten, um für

die 65 535 Speichermöglichkeiten in unserem APPLE-Computer 65 535 Adressen bilden zu können. Diese Adressen müssen wir jedoch nicht als 0-1-Kombinationen, sondern in der sehr bequemen hexadezimalen Schreibweise eingeben, die wir anschließend kennenlernen.

Doppelworte dienen uns auch in Maschinenprogrammen bei z.B. 8-Bit-Multiplikationen. Jede 8-Bit-Multiplikation ergibt ein 16-Bit-Produkt oder Doppelwort, wie in Kapitel 9 besprochen wird.

	1 Byte		1 Byte		1 Doppelwort
binär:	00010010	x	00111010	=	0000010000010100
dezimal:	18	x	54	=	1044

Binär, Hexadezimal, Dezimal, ASCII



hexadezimal

binär

Dezimalzahl "165"

ASCII (dez)

hexadezimal

binär

Buchstabe "M"

ASCII

Das Lesen und Eingeben von Zeichen beim Arbeiten mit Computern vereinfacht sich häufig durch einen Wechsel der Darstellungsart.

Beim Arbeiten mit BASIC-Programm können wir die gewohnten Schreibweisen beibehalten: Buchstaben des ABC, Dezimalzahlen, Satzzeichen und Sonderzeichen (siehe Tastenfeld des Computers).

Beim Arbeiten mit Maschinenprogrammen stehen wir dem Mikroprozessor im APPLE mit seinen 0-1-Darstellungen näher und können Informationen leichter aus binären und hexadezimalen Darstellungen entnehmen.

Zur Übertragung von Zeichen, nicht nur Zahlen, zwischen dem Computer und angeschlossenen Geräten wie Bildschirm, Drucker, Tastatur usw. wurde für 128 Zeichen ein internationaler Zeichencode vereinbart – der ASCII-Code.

Binär

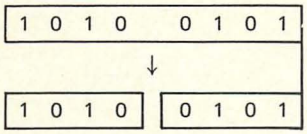
Die binäre Darstellung von Daten in Form von 0-1-Kombinationen ist eine unmittelbare Wiedergabe physikalischer Zustände im Mikroprozessor. Zur Eingabe von Maschinenprogrammen und zum Lesen ihrer Ergebnisse müssen wir jedoch nicht mit binären Darstellungen arbeiten. Mit der nachfolgend beschriebenen hexadezimalen Schreibung haben wir ein elegantes Verfahren, über Binärdaten zu sprechen, ohne 0-1-Kombinationen verwenden zu müssen.

Zu binären Darstellungen werden wir in diesem Buch nur dort zurückkehren, wo wir uns bis auf 1 Bit genau über die Wirkung unserer Maschinenbefehle Klarheit verschaffen müssen: bei arithmetischen Operationen zum Erkennen von Überläufen, zur Darstellung negativer Zahlen und zur Beschreibung der einzelnen Bits im Statusregister des Mikroprozessors.

Hexadezimal

Die 0-1-Kombinationen der 8 Bits eines Byte lassen sich hexadezimal etwa so einfach und elegant durch 2 Zeichen wiedergeben, wie Great Britain durch die Zeichen GB.

Zur hexadezimalen Darstellung eines Bytes werden dessen 8 Bits als Zusammensetzung aus zwei 4-Bit-Blöcken gesehen.



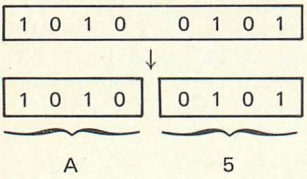
Stellt man jeden dieser 4-Bit-Blöcke durch ein Zeichen dar, dann kann man die 8 Bits eines Byte mit 2 Zeichen wiedergeben.

Hier die Tabelle der hexadezimalen Zeichen 0 . . . 9, A . . . F zur Wiedergabe der 16 möglichen 0-1-Kombinationen in jedem 4-Bit-Block:

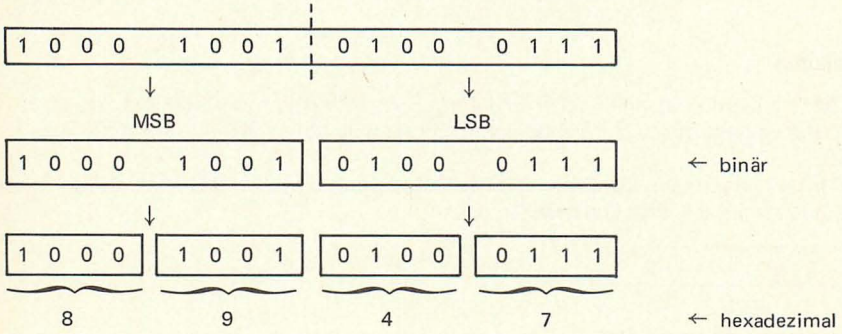
dez	bin	hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

dez = dezimal
bin = binär
hex = hexadezimal

Das eben dargestellte Byte lautet also hexadezimal A5:



Die beiden Bytes (MSB, LSB) des oben als Beispiel genannten Doppelworts werden hexadezimal als (MSB = 89, LSB = 47) wiedergegeben.

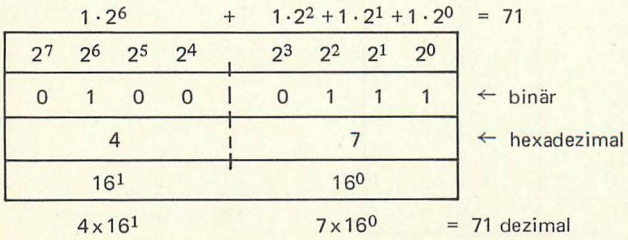


MSB	LSB
89	47

Befehle und Daten in Maschinenprogrammen werden überwiegend hexadezimal dargestellt. Anhang D zeigt die Codes der Maschinenbefehle des 6502 in hexadezimaler Form. Zur Umwandlung von dezimalen Daten in hexadezimale Schreibung oder umgekehrt, können Sie die Umwandlungstabelle in Anhang G oder ein Umwandlungsprogramm in BASIC verwenden, wie Anhang H zeigt.

Die Umwandlung Hexadezimal-Dezimal ist äußerst einfach: Statt der Gewichte $2^0, 2^1, 2^2 \ldots$ wie bei Binärzahlen, tragen Hexadezimalziffern die Gewichte $16^0, 16^1, 16^2 \ldots$

47 hex → 71 dez :



Und hier die Dezimalzahl zum Doppelwort 8947 hex:

8947 hex → 35 143 dez :

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
1	0	0	0	1	0	0	1	0	1	0	0	0	1	1	1

8 | 9 | 4 | 7 hex

8×16^3 | 9×16^2 | 4×16^1 | 7×16^0 = 35 143 dez

Auch die Umwandlung Dezimal–Hexadezimal ist einfach: Wenn wir von vierstelligen Hexadezimalzahlen ausgehen, dividieren wir die Dezimalzahl zunächst durch $16^3 = 4096$, den verbleibenden Rest teilen wir durch $16^2 = 256$, den hier verbleibenden Rest durch 16^1 und der dann noch verbleibende Rest wird unmittelbar hexadezimal dargestellt:

35 143 dez → 8947 hex :

```
35143 : 4096 = 8    Rest 2375
 2375 :  256 = 9    Rest   71
   71 :   16 = 4    Rest    7
    7 =  7   hex
```

ASCII-Codes

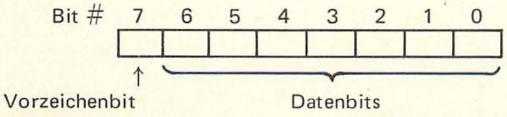
Alle über die Tastatur eingegebenen oder auf dem Bildschirm dargestellten Zeichen werden im APPLE-Computer als ASCII-Codes gespeichert. (Anhang C-3). Zum Arbeiten mit Zeichen werden wir von fest im APPLE-Computer eingespeicherten Maschinenprogrammen Gebrauch machen, die einerseits Zeichen von der Tastatur einlesen und mit ihren ASCII-Codes abspeichern und andererseits ASCII-Codes im Speicher durch ihre zugehörigen Zeichen auf dem Bildschirm darstellen.

Für Zeichendarstellungen auf dem Bildschirm werden wir 3 Darstellungsarten kennenlernen: Normal, Invers und blinkend.

Negative Binärzahlen

Zur Darstellung von Zahlen, die positiv, negativ oder Null sein können, muß der Computer eine Unterscheidungsmöglichkeit haben.

Betrachten wir ein 8-Bit-Wort, das aus einem Vorzeichen-Bit und 7 Datenbits besteht:



a) Wenn das Vorzeichen den Wert 0 enthält, soll die 7-Bit-Zahl positiv sein:

```
0 1 1 1 1 1 1 1 = +127   (64 + 32 + 16 + 8 + 4 + 2 + 1)
0 1 1 1 1 1 1 0 = +126   (64 + 32 + 16 + 8 + 4 + 2    )
0 1 1 1 1 1 0 1 = +125   (64 + 32 + 16 + 8 + 4     + 1)
0 1 1 1 1 1 0 0 = +124   (64 + 32 + 16 + 8 + 4         )
      .          .              .
      .          .              .
      .          .              .
      .          .              .
0 0 0 0 0 0 1 1 = +3     (                   + 2 + 1)
0 0 0 0 0 0 1 0 = +2     (                   + 2    )
0 0 0 0 0 0 0 1 = +1     (                       + 1)
0 0 0 0 0 0 0 0 = +0     (                           )
```

Null wird als positive Zahl dargestellt

b) Wenn das Vorzeichen den Wert 1 enthält, dann soll die 7-Bit-Zahl negativ sein:

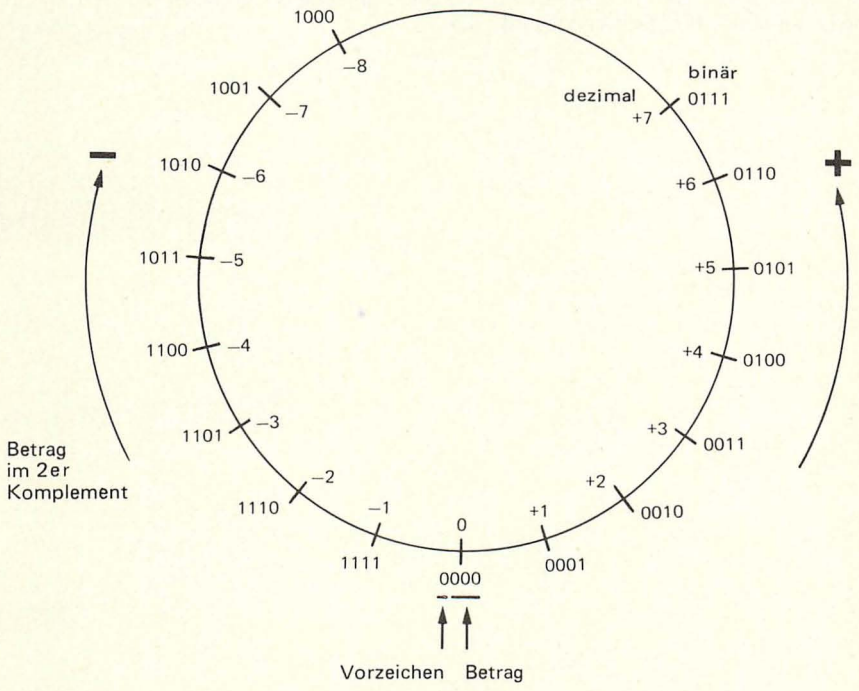
```
1 0 0 0 0 0 0 0 = – 128
1 0 0 0 0 0 0 1 = – 127
1 0 0 0 0 0 1 0 = – 126
1 0 0 0 0 0 1 1 = – 125
1 0 0 0 0 1 0 0 = – 124
      .
      .
      .
      .
1 1 1 1 1 0 1 1 = – 5
1 1 1 1 1 1 0 0 = – 4
1 1 1 1 1 1 0 1 = – 3
1 1 1 1 1 1 1 0 = – 2
1 1 1 1 1 1 1 1 = – 1
```

Damit könnten wir ein 8-Bit-Wort nicht mehr für die Darstellung der positiven Zahlen 0 . . . 255, sondern nur zur Darstellung der Dezimalzahlen –128 . . . +127 verwenden:

8-Bit-Wort:

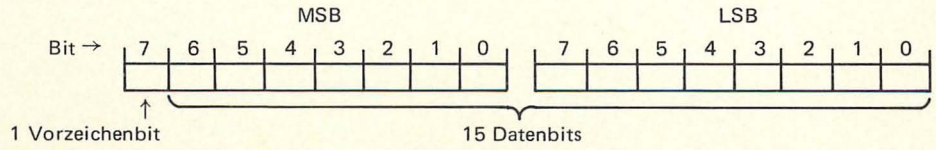
```
0 0 0 0 0 0 0 0 = +0   ┐
0 0 0 0 0 0 0 1 = +1   │
      .                │
      .                │ positiv
      .                │
0 1 1 1 1 1 1 0 = +126  │
0 1 1 1 1 1 1 1 = +127  ┘
1 0 0 0 0 0 0 0 = – 128 ┐
1 0 0 0 0 0 0 1 = – 127 │
1 0 0 0 0 0 1 0 = – 126 │
      .                │ negativ
      .                │
      .                │
1 1 1 1 1 1 1 0 = – 2   │
1 1 1 1 1 1 1 1 = – 1   ┘
```

Die binäre Darstellung negativer Zahlen stellt das 2er-Komplement ihrer positiven Entsprechung dar.



positiv	2er-Komplement		negativ
0101	0101		1011
	↓	invertieren	
	1010		
	+ 1	"1" addieren	
	1011		

Bei Doppelworten gilt das höchstwertige Bit im höchstwertigen Byte als Vorzeichenbit:



Negative Hexadezimalzahlen

Da Hexadezimalzahlen lediglich eine bequemere Verschlüsselung von Binärzahlen darstellen, gelten hier die Aussagen, die wir bereits über negative Binärzahlen gemacht haben.

Wir werden in Maschinenprogrammen zur Bildung von Schleifen und für Verzweigungen von positiven und negativen Hexadezimalzahlen Gebrauch machen, die in Anhang B tabellarisch zusammengestellt sind.

3
Alles über Speicher

Einführung

Alle Programme müssen vor ihrem Ablauf im Speicher abgelegt werden. Während uns in BASIC ein Übersetzungsprogramm, der Interpreter, die Programmablage im Speicher abnimmt, müssen wir bei Maschinenprogrammen selbst die Speicheradressen angeben, unter denen die Maschinenbefehle und Daten stehen sollen, und auch selbst für die Programmablage sorgen.

Die Ablage von Maschinenprogrammen im Speicher ist nicht schwierig – wir müssen nur die Speicherbereiche kennen, in denen das möglich ist.

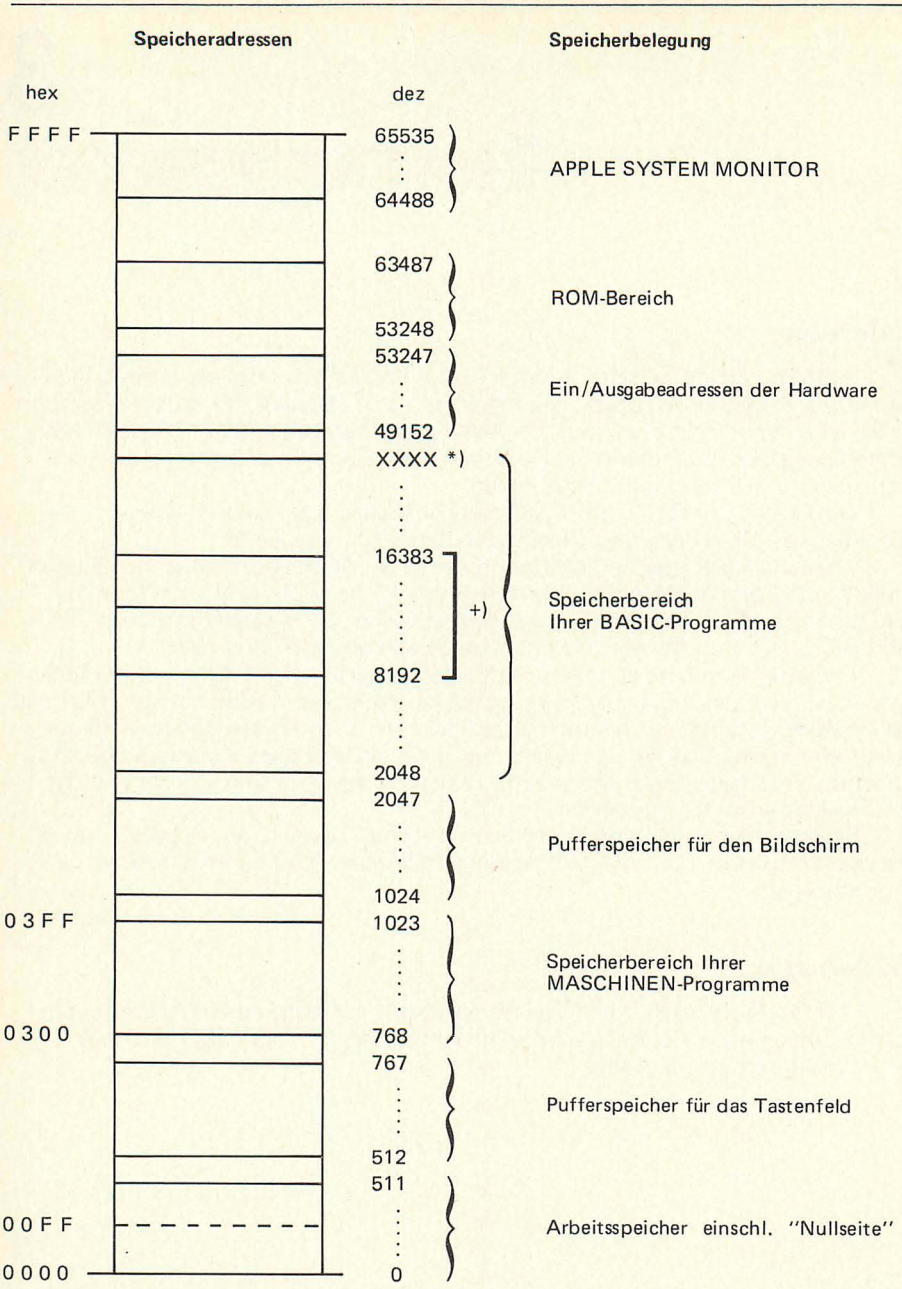
Auskunft über Speicherbereiche, in denen wir Maschinenprogramme ablegen können, und über die Adressen dieser Speicherbereiche erfahren wir aus "Speicherauszügen" oder "Memory Maps", die beispielsweise im Benutzerhandbuch des APPLE-Computers zu finden sind und nachfolgend besprochen werden.

Für seine Kommunikation mit dem Speicher verfügt der Mikroprozessor über eigene Zwischenspeichermöglichkeiten, genannt Register und Akkumulator. Während wir in BASIC-Programmen nie von diesen Speichern erfahren, stehen uns in Maschinensprache Befehle zur Verfügung, mit denen wir planvoll diese Speichermöglichkeiten zum Zwischenspeichern von Daten, zur Bildung von Adressen und zur Abfrage von Ergebnissen einsetzen können.

Programmieren in Maschinensprache eröffnet uns auch den Zugang zu einer besonders eleganten Form der Zwischenspeicherung von Daten im sogenannten Stapelspeicher.

Speicherauszug

Auf der Suche nach freien Speicheradressen zur Ablage unserer Machinenprogramme finden wir in "Speicherauszügen" oder "Memory Maps" des Reference Manual die benötigte Auskunft.



*) abhängig von Speicherkapazität des Computers:
16K: XXXX = 16 383
32K: XXXX = 32 767
48K: XXXX = 49 151

+) Speicherbereich für hochauflösende Graphiken

Der Blick auf einen derartigen Speicherauszug zeigt uns, daß die 65535 möglichen Speicheradressen des APPLE teilweise bereits vergeben sind. Neben den Programmen des Anwenders stehen in diesem Speicher auch Programme, die der APPLE zur Abwicklung seiner Aufgaben benötigt – die sogenannten Betriebsprogramme.

Für Maschinenprogramme stehen uns der Adreßbereich 768 . . . 1023 dez (oder 0300 . . . 03FF hex) mit 255 Speicherplätzen und, wenn dies nicht genügen sollte, Speicheradressen im für BASIC-Programme reservierten Speicherbereich zur Verfügung.

Von besonderer Bedeutung für Maschinenprogramme ist auch der Speicherbereich mit den Adressen 0 . . . 255 dez (oder 0000 . . . 00FF hex) mit 256 Speicherstellen. Auch hier stehen uns einige dieser Adressen für Maschinenprogramme zur Verfügung. Ihr Vorteil liegt in der Möglichkeit, besonders kurze und schnelle Maschinenbefehle anzuwenden, in denen nur das LSB 00 . . . FF hex der Adresse angegeben werden muß und die daher um 1 Byte kürzer sind. Dieser Speicherbereich trägt auch die besondere Bezeichnung "Page Zero" oder "Nullseite".

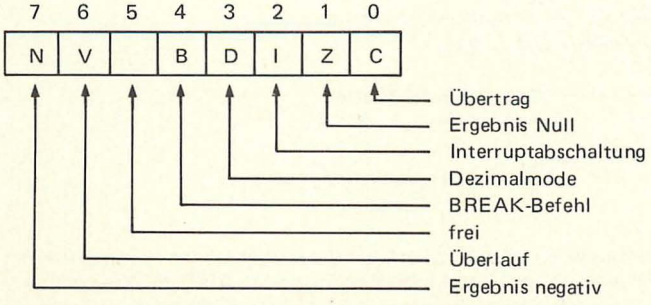
Von besonderer Bedeutung für Maschinenprogramme ist auch der anschließende Speicherbereich mit den Adressen 256 . . . 511 dez (oder 0100 . . . 01FF hex) und ebenfalls 256 Speicherstellen. Dieser Bereich bildet den Stapelspeicher des Mikroprozessors 6502, den wir während des Ablaufs von Maschinenprogrammen zur zwischenzeitlichen Speicherung von Daten benutzen werden.

Akkumulator und Register

Der Mikroprozessor 6502 im APPLE-Computer verfügt über drei Zwischenspeicher: den Akkumulator A und die Indexregister X und Y. Alle Speicher-Speicher-Übertragungen von Daten müssen über eines dieser Register laufen.

Der Akkumulator A und die Register X, Y können wie andere Speicherplätze ein 8-Bit-Wort speichern. Der Befehlssatz des Mikroprozessors 6502 enthält nicht nur Befehle für Datenübertragungen von und zu diesen Registern, sondern auch Befehle zur Veränderung ihrer Inhalte. Mit wenigen Ausnahmen kann der 6502 arithmetische und logische Operationen nur an Inhalten seiner Zwischenspeicher ausführen.

Neben diesen Registern, deren Funktion in den folgenden Maschinenprogrammen sichtbar wird, verfügt der Mikroprozessor 6502 über ein Statusregister zur Anzeige des Ergebnisses von Operationen:



und ein Register für den Zeiger zum Stapelspeicher, den wir anschließend darstellen werden.

Lesen und Setzen der Register

Der Inhalt aller Speicher, über die der Mikroprozessor 6502 im APPLE-Computer verfügt:

A	Akkumulator
X	Indexregister X
Y	Indexregister Y
P	Statusregister
S	Register für den Zeiger zum Stapelspeicher

läßt sich durch Eingabe von CTRL-E und RETURN auf dem Bildschirm darstellen. Unmittelbar nach dem Einschalten des APPLE-Computers haben die Register folgenden Inhalt:

Ihre Eingaben	**Bildschirmantwort**
CTRL E	*
RETURN	A=FF X=FF Y=FF P=00 S=FF *■

Der Inhalt der Register A, X, Y, P, S kann anschließend durch Eingabe eines Doppelpunkts, dem die neuen Registerinhalte in hexadezimaler Schreibung und jeweils mit einem Zwischenraum folgen, geändert werden:

```
*
A=FF X=FF Y=FF P=00 S=FF
*:88 FF 80 33 40
*■
  A X Y P S
```

von Ihnen eingegebene Registerinhalte

Um zu prüfen, ob die Inhalte tatsächlich geändert wurden, geben wir wieder CTRL-E und RETURN ein:

```
A=FF X=FF Y=FF P=00 S=FF
*:88 FF 80 33 40
*
A=88 X=FF Y=80 P=33 S=40
*■
```

ursprüngliche Inhalte

Änderungen

CTRL-E und RETURN

Anzeige der geänderten Inhalte

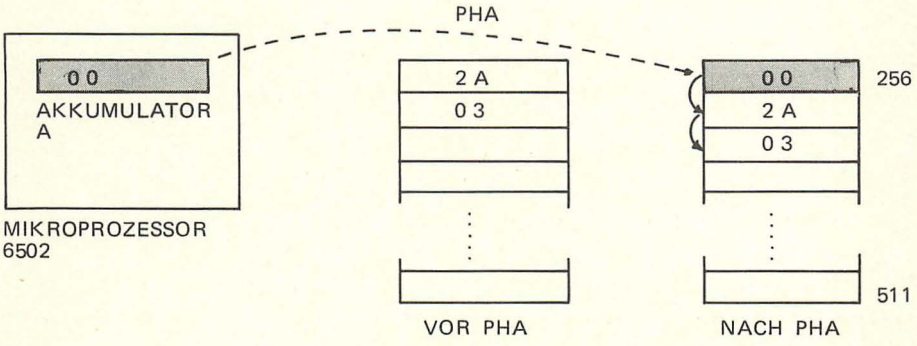
Diese Inspektionsmöglichkeit ist besonders bei der Fehlersuche in Maschinenprogrammen nützlich. Wir werden im Zusammenhang mit dem APPLE-Monitor die Möglichkeit kennenlernen, ein Maschinenprogramm Befehl für Befehl ausführen zu lassen und jeweils die Inhalte der Register, A, X, Y, P, S auf dem Bilschirm darzustellen und zu überprüfen.

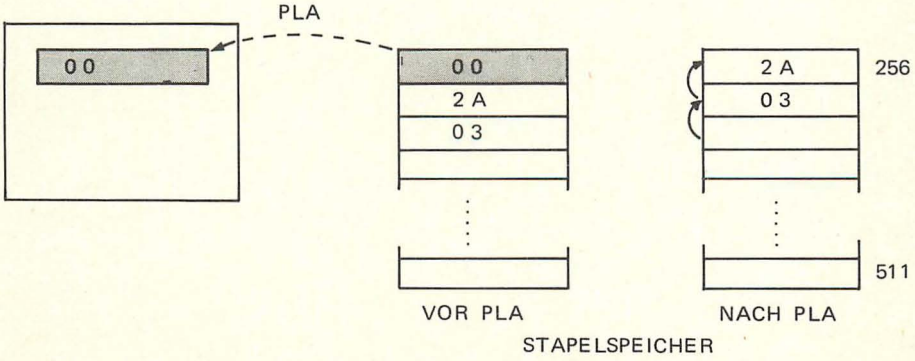
Stapelspeicher

Zur vorübergehenden Ablage von Daten stehen dem Mikroprozessor 6502 außer seinen Registern X, Y und dem Akkumulator A auch der Speicherbereich mit den Adressen 0100 . . . 01FF hex bzw. 256 . . . 511 dez zur Verfügung. Dieser Speicherbereich wird der "Stapelspeicher" oder "Page One" genannt.

Die Ablage von Daten im Stapelspeicher erinnert an die Funktion des Tablettspenders in Kantinen: Wird ein Tablett aufgelegt, dann sinkt der ganze Stapel nach unten, wird ein Tablett abgenommen, dann steigt der ganze Stapel um eine Tablettdicke nach oben. Das zuletzt auf den Stapel gelegte Tablett wird auch als erstes wieder abgenommen.

Der Stapelspeicher im APPLE-Computer kann bis zu 256 Bytes aufnehmen. Mit den Maschinenbefehlen PHA bzw. PLA können Daten vom Akkumulator auf den Stapelspeicher abgeladen bzw. vom Stapelspeicher in den Akkumulator zurückgeholt werden.





Daneben wird der Stapelspeicher bei Sprüngen in Unterprogramme zur Ablage der Rücksprungadresse verwendet. Beim Arbeiten mit dem Stapelspeicher müssen

daher sorgfältig Überscheidungen von Befehlen zur Ablage von Daten und von Befehlen zum Sprung in Unterprogramme vermieden werden.

4

Alles über Maschinenbefehle

Einführung

Alle Befehle, ob in BASIC oder Maschinensprache, nehmen letztlich die einzige Form an, in der der Mikroprozessor sie ausführen kann: sie stehen im Speicher als 0-1-Folge von 8 Bits. Der Weg dorthin ist in BASIC länger als in Maschinensprache – daher laufen BASIC-Programme erheblich langsamer ab.

Da alle 8-Bit-Worte im Speicher mit ihren 0-1-Folgen gleich aussehen, muß dem Mikroprozessor mitgeteilt werden, unter welcher Speicheradresse der erste Befehl des Maschinenprogramms steht. Hat der Mikroprozessor durch diesen Hinweis das erste 8-Bit-Wort richtig als Befehl verstanden, dann weiß er alle weiteren 8-Bit-Worte richtig als Befehle, Adressen oder Daten zu interpretieren.

Hier ein Beispiel: Dem Mikroprozessor wird mitgeteilt, daß unter Adresse 0300 hex (oder 768 dez) der erste Maschinenbefehl des Maschinenprogramms steht. Aus dem Binärmuster 10101101 oder AD hex in dieser Speicherstelle erkennt der Mikroprozessor, daß es sich um einen Ladebefehl handelt, zu dem noch eine Adreßangabe gehört. Daher interpretiert der Mikroprozessor den Inhalt der folgenden beiden Bytes (Adressen 0301 und 0302) als Adreßangabe. Mit dieser Adreßangabe spricht er die Speicherstelle mit der Adresse 03F0 an und interpretiert deren Inhalt richtig als Daten, in diesem Beispiel als die Zahl 13 hex oder 910 dez.

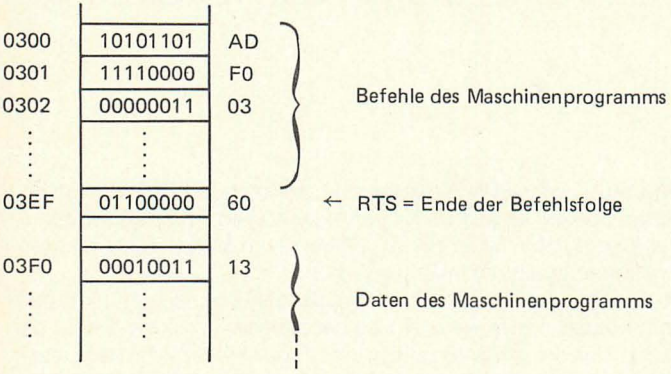
Speicheradresse		Speicherinhalt		Bedeutung
dez	hex	bin	hex	
768	0300	10101101	AD	**Maschinenbefehl** LDA 03F0
769	0301	11110000	F0	"Lade Daten von Adresse 03F0 in
770	0302	00000011	03	Akkumulator A"
⋮	⋮	⋮		
1008	03F0	00010011	13	**Daten:** 13 hex = 19 dez

Den Hinweis auf die Speicheradresse, unter der der erste Befehl des Maschinenprogramms steht, geben wir dem Mikroprozessor mit dem BASIC-Befehl CALL. Angenommen, dieser Befehl stehe in Programmzeile 800 unseres BASIC-Programms, dann hätten wir in unserem Beispiel:

```
800 CALL 768
```

Das Ende des Maschinenprogramms, d.h. das Ende der Folge von Maschinenbefehlen im Speicher, wird dem Mikroprozessor mit dem Maschinenbefehl RTS (RETURN FROM SUBROUTINE) mitgeteilt. Dieser Maschinenbefehl RTS steht als

Binärwort 0110000 im Speicher und wird, hexadezimal abgekürzt, mit dem Code 60 dargestellt. Alle nach diesem Maschinenbefehl RTS im Speicher stehenden 8-Bit-Worte werden vom Mikroprozessor nicht mehr als Maschinenbefehle interpretiert. Für gewöhnlich findet man daher Daten, von denen ein Maschinenprogramm Gebrauch macht, am Ende der eigentlichen Befehlsfolge des Maschinenprogramms, d.h. nach RTS.



Befehlsdarstellung

In der Literatur und bei der Eingabe über das Tastenfeld finden wir Maschinenbefehle in einer der folgenden Darstellungsformen:

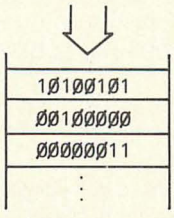
- mnemonisch (Assembler-Form)
- hexadezimal
- dezimal

Im Speicher unseres Computers stehen alle Befehle – gleichgültig, wie sie bei der Eingabe aussahen – in binärer Form, d.h. als 0-1-Folge.

Zur Programmierung in Maschinensprache müssen wir jedoch nicht derart "hautnah" Abläufe im Mikroprozessor vor Augen haben, so daß wir binären Darstellungen von Maschinenbefehlen so gut wie nie begegnen.

Hier als Beispiel möglicher Befehlsdarstellungen der Befehl "LDA oper = Lade Akkumulator A mit Inhalt des Speichers, dessen Adresse oper ist".

Befehlsdarstellung		**Anwendung**
allgemein:	LDA oper	Darstellung der Befehlsform
mnemonisch:	LDA Ø32Ø	Beim Schreiben der Programme
hexadezimal:	A5 Ø3 2Ø	Beim Eingeben der Programme
dezimal:	165 8ØØ	Im BASIC-Befehl POKE



A5 LDA Ø32Ø *)
2Ø
Ø3

*) im Speicher wird die Adresse Ø32Ø in der Reihenfolge: erst LSB, dann MSB abgelegt!

Beim Arbeiten in BASIC müssen wir von unserer gewohnten dezimalen Darstellung von Zahlen nicht abweichen. BASIC-Befehle erwarten dezimale Zahleneingaben. Typisches Beispiel für die Eingabe von Maschinenbefehlen mit Hilfe von BASIC-Befehlen ist POKE:

```
POKE adresse, datum
```

Der BASIC-Befehl POKE erwartet für Adresse und Inhalt der angesprochenen Speicherstelle dezimale Angaben. Im nachfolgend beschriebenen Betriebssystem BBS, das der Eingabe von Maschinenprogrammen dient, ist daher eine Hexadezimal-Dezimal-Umwandlung enthalten, um Maschinenbefehle nach der Liste in Anhang D in hexadezimaler Form eingeben und dem BASIC-Befehl POKE in dezimaler Form übergeben zu können.

Adressierungsarten

Zu allen Maschinenbefehlen des Mikroprozessors 6502 in Anhang D finden wir eine Angabe zur "Adressierungsart". Folgende Adressierungsarten werden genannt:

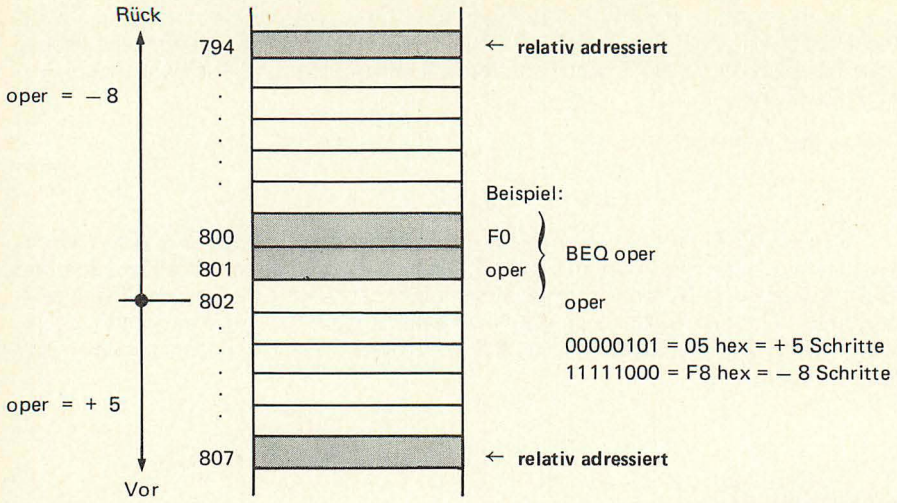
Absolut
Relativ
Unmittelbar
Indirekt
Zero Page
Indiziert

Bei vielen Maschinenbefehlen liegt die Adressierungsart fest (implizit, relativ). Bei einigen Maschinenbefehlen kann unter mehreren Adressierungsarten gewählt werden: Zum gleichen Maschinenbefehl, z.B. LDA, gibt es dann mehrere Codes (A1, A5, A9, B1 . . . im Fall von LDA) zur Unterscheidung der Adressierungsarten.

Relativ

Diese Adressierungsart tritt nur bei Verzweigungsbefehlen wie BEQ, BNE, BNI . . . auf (B = BRANCH, Verzweigen).

Bei diesen Verzweigungsbefehlen gibt man in Form einer Hexadezimalzahl an, um wieviele Speicheradressen im Verzweigungsfall vor- oder zurückgesprungen werden soll.



Die Verzweigungsrichtung wird durch das höchste Bit im 8-Bit-Operanden des Maschinenbefehls angezeigt: Der Wert 1 bedeutet eine negative Hexadezimalzahl und Rücksprung, der Wert 0 eine positive Hexadezimalzahl und Vorsprung. Die darstellbaren Wert +127 . . . –128 eines 8-Bit-Worts mit Vorzeichen beschränken die maximale Sprungweite relativer Adressierungen. Den Hexadezimalwert der Verzweigungsschritte entnehmen wir einer Tabelle in Anhang B, die hier auszugsweise dargestellt ist:

VORVERZWEIGUNG

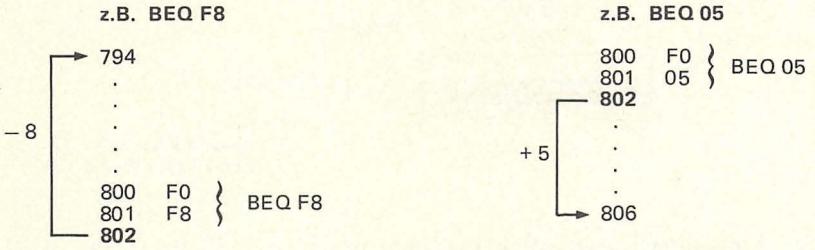
Verzweigungsschritte	
dez	hex
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	08
9	09
10	0A
11	0B
12	0C
13	0D
14	0E
15	0F
16	10
.	.
.	.
.	.

RÜCKVERZWEIGUNG

Verzweigungsschritte	
dez	hex
1	FF
2	FE
3	FD
4	FC
5	FB
6	FA
7	F9
8	F8
9	F7
10	F6
11	F5
12	F4
13	F3
14	F2
15	F1
16	F0
.	.
.	.
.	.

"Relative Adressierung" bedeutet also die Adressierung von Speicheradressen "relativ" zur Speicheradresse des Verzweigungsbefehls.

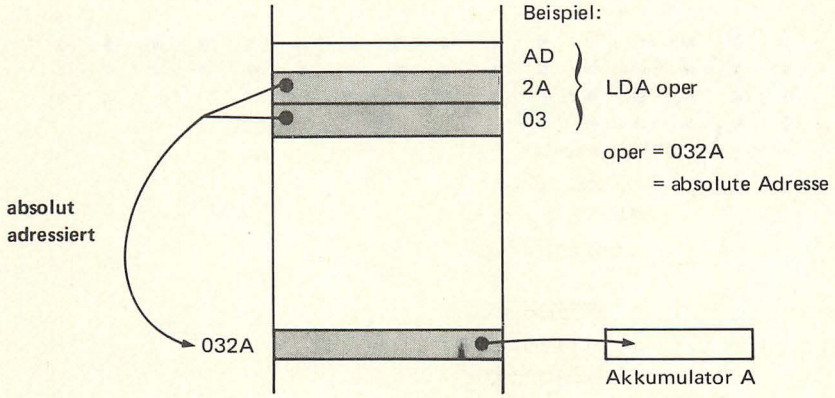
Zur Bestimmung der Sprungweite ist zu beachten, daß der Programmzähler im Mikroprozessor während der Bearbeitung des Verzweigungsbefehls bereits die Speicheradresse des nachfolgenden Befehls geladen hat. Die Sprungweite muß also von der Speicheradresse des nachfolgenden Befehls aus gerechnet werden:



Absolut

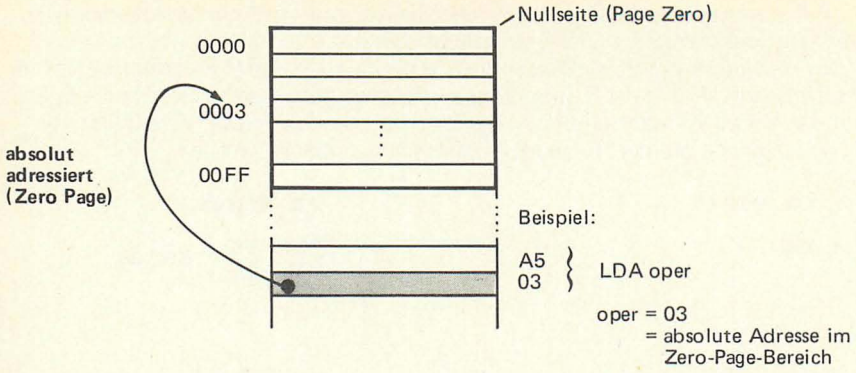
Man kann in Befehle für Speicherzugriffe die "absolute" oder "tatsächliche" Adresse des gewünschten Speicherworts eintragen. Man spricht dann von "absoluter Adressierung".

Hier ein Beispiel: Wird der Ladebefehl LDA mit dem Operationscode AD benutzt, dann erwartet der Mikroprozessor absolute Adressierung. Der Mikroprozessor erwartet nach dem Byte AD zwei Bytes mit der absoluten Adresse desjenigen Speicherworts, das in den Akkumulator A geladen werden soll. (Die absolute Adresse steht in der Reihenfolge: erst LSB, dann MSB nach dem Maschinenbefehl AD).



Zero Page

Die Adressierungsart "Zero Page" ist ebenfalls eine absolute Adressierungsart, nur sind die Adressen auf den Bereich 0000 . . . 00FF hex beschränkt. Dieser Bereich heißt "Page Zero oder "Nullseite".



In Page-Zero-Adressierung entfällt das höchstwertige Byte MSB = 00 hex in der Adreßangabe des Maschinenbefehls. Derartige Maschinenbefehle belegen daher nicht nur weniger Speicherplatz, sondern laufen auch schneller ab. Bei ihrer Benutzung ist jedoch Vorsicht geboten, da wichtige Betriebsprogramme des APPLE-Computers ebenfalls mit Page-Zero-Adressen arbeiten und durch Überschneidungen Funktionsstörungen möglich sind.

Beachten Sie daher die Angaben zur Speicherbelegung der Nullseite in Ihrem Benutzerhandbuch. Hier ein Auszug aus den Angaben, die Sie in Anhang F dieses Buchs finden:

Speicherbelegung der Nullseite durch den APPLE MONITOR

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000																
001																●
002	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
003	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
004	●	●	●	●	●	●	●	●	●	●					●	●
005	●	●	●	●	●	●										
006																
007																
008																
009																
00A																
00B																
00C																
00D																
00E																
00F																

Implizit

Bei Maschinenbefehlen mit "impliziter Adressierung" erkennt der Mikroprozessor schon beim Lesen des Befehlscodes,welcher Speicherplatz anzusprechen ist. Ohne Ausnahme handelt es sich um Speicherplätze innerhalb des Mikroprozessors.

Typische Beispiele sind DEX, DEY: Dekrementiere das X-Register, Y-Register. TAX, TXA: Transferiere den Inhalt des Akkumulators A in das X-Register, transferiere den Inhalt des X-Registers in den Akkumulator A.

Alle Befehle mit impliziter Adressierung sind 1-Byte-Befehle.

Unmittelbar

Maschinenbefehle mit "unmittelbarer Adressierung" benutzen eine Konstante, die in den Bytes unmittelbar nach dem Operationscode des Maschinenbefehls im Speicher steht.

Diese Adressierungsart kann auch als eine Erweiterung der impliziten Adressierung aufgefaßt werden: Der Befehlscode enthält bereits die Information, welcher Speicher im Mikroprozessor angesprochen ist. Die im Maschinenbefehl enthaltene Konstante wird für die Operation mit dieser Speicherstelle benutzt.

Typische Beispiele sind CPX #oper: Vergleiche den Inhalt des X-Registers mit der Konstanten oper im Maschinenbefehl, LDA #oper: Lade die Konstante oper in den Akkumulator A.

Alle Befehle mit unmittelbarer Adressierung sind 2-Byte-Befehle.

Indiziert

Maschinenbefehle mit "indizierter Adressierung" gestatten, eine Folge von Speicheradressen für den Zugriff auf aufeinanderfolgende Speicherworte zu bilden.

Das Prinzip der indizierten Adressierung ist sehr einfach: Zu einer Anfangsadresse wird der stetig inkrementierte (oder dekrementierte) Inhalt des X- oder auch Y-Registers addiert. X- bzw. Y-Register bilden einen "Index" oder "Zeiger" und heißen daher auch häufig Indexregister.

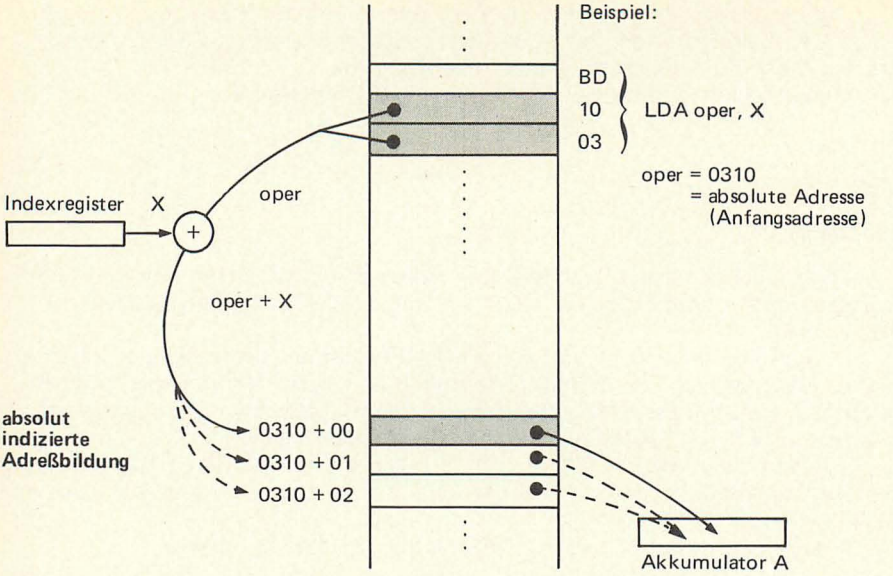
Zur indizierten Adressierung gibt es folgende Möglichkeiten:

- Absolut-indizierte-Adressierung
- Zero-Page-Indizierung
- Indiziert-indirekte-Adressierung
- Indirekt-indizierte-Adressierung

Absolut-indizierte-Adressierung

In dieser Adressierungsart enthält der Maschinenbefehl eine absolute Adresse – die tatsächliche Adresse eines gewünschten Speicherworts – die mit dem Inhalt des X- oder Y-Registers indiziert wird. Jede Speicheradresse von 0000 . . . FFFF hex ist als Anfangsadresse zugelassen.

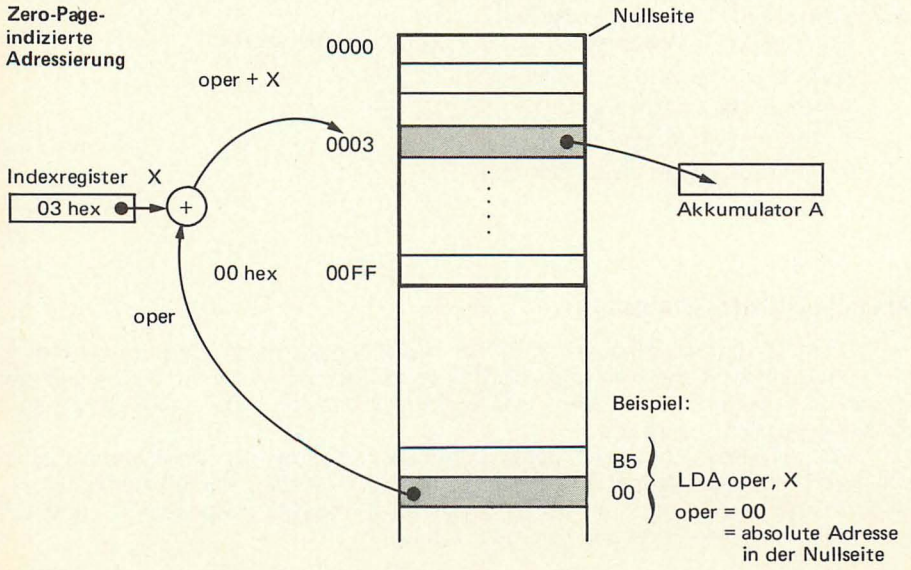
Hier ein Beispiel: Mit dem Befehl LDA oper, X werden die Speicherworte mit den Speicheradressen oper + X in den Akkumulator A gelesen. Vor jeder erneuten Ausführung von LDA oper, X wird der Inhalt des X-Registers beispielsweise inkrementiert, um so eine Folge von Adressen zu bilden:



Zero-Page-indizierte Adressierung

Maschinenbefehle mit dieser Adressierungsart arbeiten mit absoluten Adressen im Bereich der Nullseite, also 0000 . . . 00FF hex, die mit dem Inhalt des X-Registers indiziert werden. (Ausgenommen die Befehle LDX = Lade das X-Register und STX = Übertrage vom X-Register in den Speicher).

Durch Fortfall des höchstwertigen Adreßbytes MSB = 00 hex belegen diese Befehle nur zwei Bytes.



Indiziert-Indirekte-Adressierung

Bei dieser Adressierungsart bildet der Maschinenbefehl mit Hilfe des X-Registers einen Zeiger in den Speicherbereich 0000 . . . 00FF hex (Page Zero), in dem sich in zwei aufeinanderfolgenden Bytes die Speicheradresse für den beabsichtigten Speicherzugriff befindet.

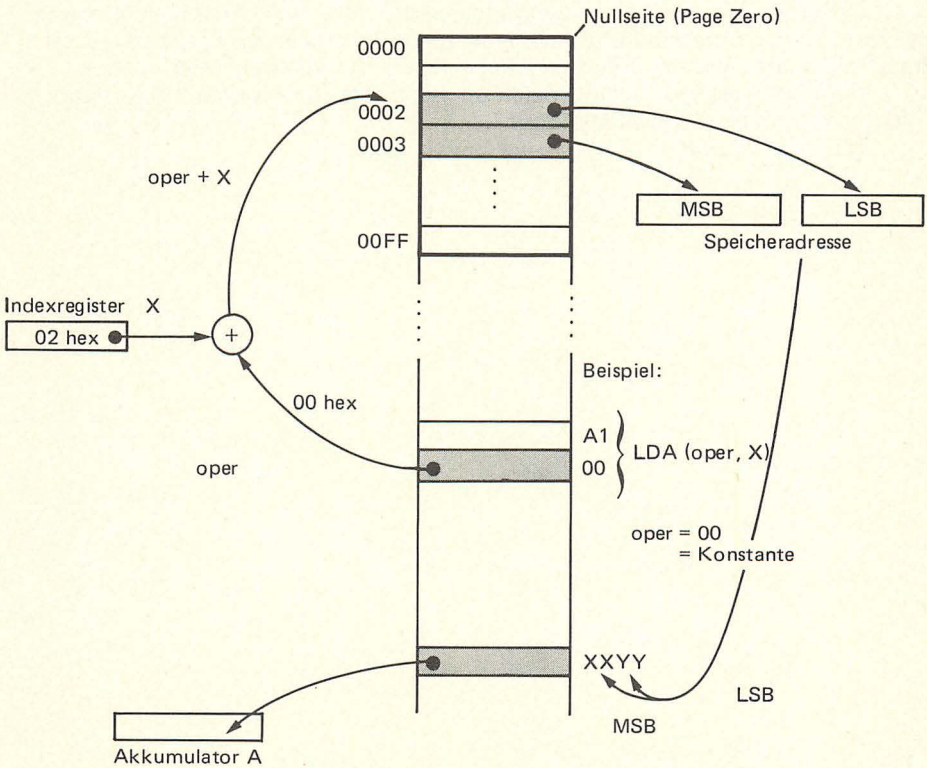
Die Adresse ist also nicht direkt im Maschinenbefehl, sondern "indirekt" in 2 Bytes des Speicherbereichs 00 . . . 00FF hex gegeben.

Das X-Register zeigt lediglich auf die Speicherstelle im Speicherbereich 0000 . . . 00FF hex, in der die gewünschte Speicheradresse steht, d.h. es "indiziert" die Adressierung.

Zur "Nullpunktverschiebung" des Zeigers in den Page-Zero-Bereich enthält der Maschinenbefehl eine frei wählbare 1-Byte-Konstante, deren Wert vor Befehlsausführung zum Inhalt des X-Registers addiert wird. Ein auftretender Übertrag wird unterdrückt.

Die gewünschte Speicheradresse steht in 2 Bytes: demjenigen, auf das der Zeiger "oper + X" weist, und in dem unmittelbar darauf folgenden Byte.

Hier ein Beispiel:



indiziert-indirekte Adressierung

Als erstes wird der Inhalt des X-Registers 02 hex zum Wert 00 hex der Konstanten oper addiert: 02 + 00 = 02. Das Ergebnis ist die Adresse 0002 hex im Page-Zero-Bereich, deren Inhalt den niederwertigen Adreßteil LSB der gewünschten Speicheradresse darstellt. Der höherwertige Adreßteil MSB der gewünschten Speicheradresse steht im Page-Zero-Bereich unter der nachfolgenden Adresse 0003 hex.

Mit MSB, LSB ist diejenige Speicheradresse gefunden, deren Inhalt anschließend in den Akkumulator A zu übertragen ist.

Nachteil der indiziert-indirekten Adressierung ist ihr großer Speicherbedarf im Bereich der Nullseite 0000 . . . 00FF hex. Vorteil dieser wie auch der folgenden Adressierungsart ist, daß die Anfangsadresse des indiziert durchlaufenden Adreßbereichs programmiert verändert werden kann.

Indirekt-indizierte Adressierung

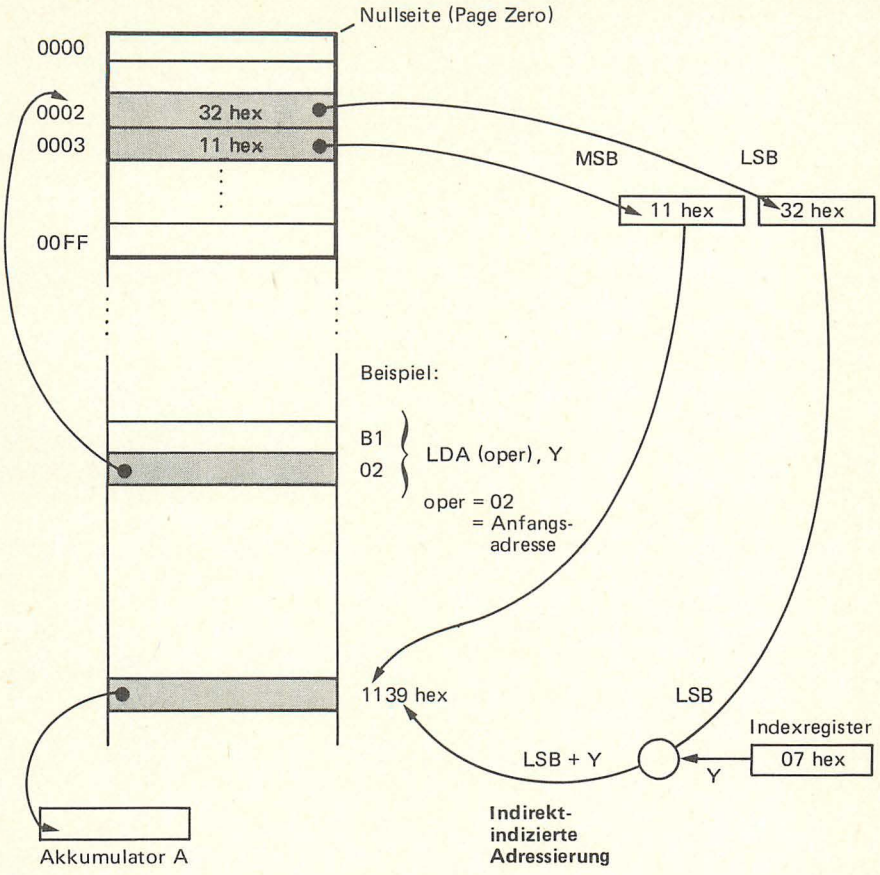
In dieser Adressierungsart kann nicht nur der Adreßzeiger im Wertebereich 0 . . . 255, sondern auch die Anfangsadresse des überstrichenen Speicherbereichs verändert werden.

Maschinenbefehle in dieser Adressierungsart bestehen aus nur 2 Bytes: Das 1. Byte enthält den Operationscode des Befehls, das 2. Byte enthält eine Adresse aus dem Speicherbereich 0000 . . . 00FF hex der Nullseite.

Die im Page-Zero-Bereich adressierte Speicherstelle enthält das niedrigstwertige Byte LSB der Anfangsadresse eines indiziert zu durchlaufenen Speicherbereichs. Im unmittelbar folgenden Byte steht der höchstwertige Adreßteil MSB.

Der Adreßzeiger wird durch Addition des Inhalts von Register Y zum niedrigstwertigen Adreßteil LSB gebildet.

Hier ein Beispiel:



Zuerst liest der Mikroprozessor den niedrigstwertigen Adreßteil LSB = 32 hex aus der Speicherstelle 0002. Dann wird zu diesem Wert der Wert 07 hex im Indexregister Y addiert. Die Summe 32 + 7 = 39 bildet den niedrigstwertigen Adreßteil der indizierten Speicheradresse. Schließlich wird der höchstwertige Adreßteil MSB = 11 hex aus Speicherstelle 0003 gelesen und als höchstwertiger Adreßteil der indizierten Speicheradresse verwendet. Der Akkumulator A wird daher mit dem Inhalt der Speicherstelle 1139 hex geladen.

Der Vorteil dieser Adressierungsart ist die frei vom Programm veränderbare und im Zero-Page-Bereich abgelegte Anfangsadresse desjenigen Speicherbereichs, der indiziert durchlaufen werden soll. Da der Adreßzeiger, hier mit den 8 Bits des Y-Registers gebildet, nur die Werte 0 . . . 256 durchlaufen kann, wären bei fester Anfangsadresse die indiziert durchlaufbaren Speicherbereiche auf 256 Speicherplätze beschränkt. Indirekt-indizierte Adressierung gestattet dagegen das indizierte Durchlaufen beliebig großer Adreßbereiche durch Verändern der Anfangsadresse.

5

BBS-Maschinenprogramme mit BASIC eingeben

Einführung

Wir können Maschinenprogramme mit dem BASIC-Befehl POKE in den Speicher des APPLE-Computers laden, mit CALL zum Ablauf bringen und mit PEEK nach ihren Ergebnissen abfragen.

Hier ein BASIC-Programm, das mit POKE-Befehlen die Dezimalcodes 169, 19, 141, 37, 3, 96 eines Maschinenprogramms unter den Speicheradressen 768 . . . 773 ablegt und anschließend in einer Programmschleife die Speicherinhalte mit PEEK liest und auf dem Bildschirm darstellt:

BASIC-PROGRAMM:

```
100 REM * BILDSCHIRM LÖSCHEN
110 HOME

200 REM * CODES MIT POKE ABSPEICHERN
210 POKE 768, 169
220 POKE 769, 19
230 POKE 770, 141
240 POKE 771, 37
250 POKE 772, 3
260 POKE 773, 96

300 REM * CODES MIT PEEK LESEN
310 FOR X = 768 TO 773
320 PRINT PEEK (X)
330 NEXT X
```

Speicheradressen: 768 . . . 773
Speicherinhalte: 169, 19, 141, 37, 3, 96 (Codes eines Maschinenprogramms)

BILDSCHIRMMELDUNGEN DES BASIC-PROGRAMMS:

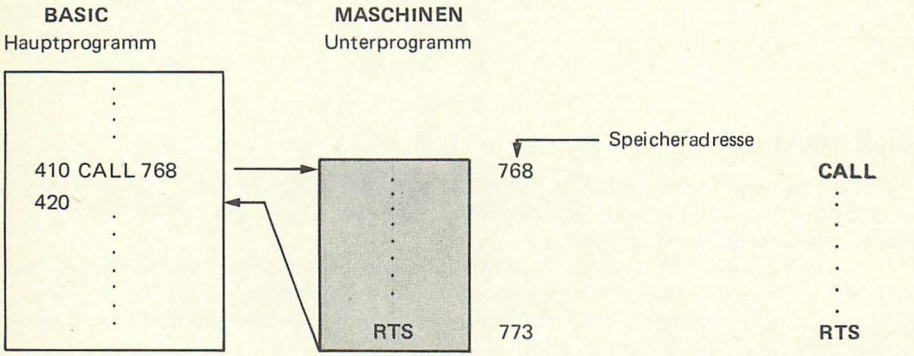
```
169
19
141
37
3
96
]■
```

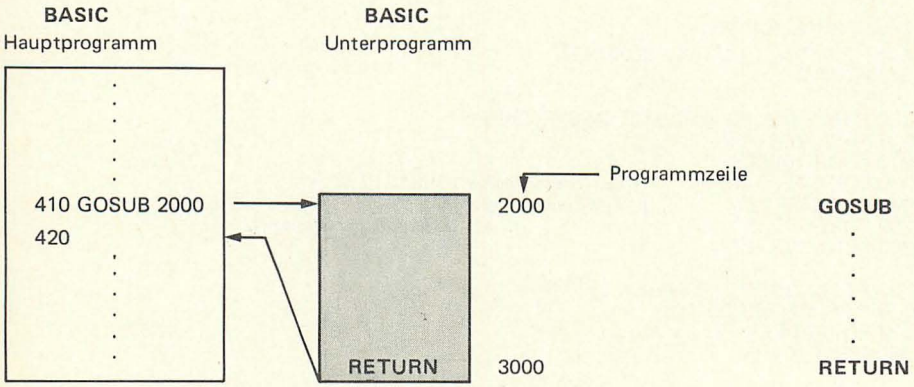
gelesene Speicherinhalte

Das Maschinenprogramm steht jetzt im Speicher des APPLE-Computers und ist ablaufbereit. Mit dem BASIC-Befehl:

```
410 CALL 768
```

können wir in dieses Maschinenprogramm wie in ein BASIC-Unterprogramm springen und seinen Ablauf auslösen. Der Befehl CALL hat große Ähnlichkeit mit dem Befehl GOSUB zum Sprung in ein BASIC-Unterprogramm:





Für die Rückkehr in das aufrufende BASIC-Programm sorgt der letzte Befehl im Maschinenprogramm mit der Bezeichnung RTS. (Den Code 60 hex oder 90 dez für diesen Befehl finden Sie in Anhang D).

Wir erweitern unser BASIC-Programm um Zeile 410 für einen CALL-Befehl und Zeile 510 für einen PEEK-Befehl, um das Ergebnis des Maschinenprogrammlaufs sichtbar zu machen. Das Maschinenprogramm soll den Zahlenwert 19 dez in die Speicherstelle mit der Adresse 805 dez laden:

```
100 REM * BILDSCHIRM LÖSCHEN
110 HOME

200 REM * CODES MIT POKE ABSPEICHERN
210 POKE 768, 169
220 POKE 769, 19
230 POKE 770, 141
240 POKE 771, 37
250 POKE 772, 3
260 POKE 773, 96

300 REM * CODES MIT PEEK LESEN
310 FOR X = 768 TO 773
320 PRINT PEEK (X)
330 NEXT X

400 REM * MASCHINENPROGRAMM STARTEN
410 CALL 768

500 REM * ERGEBNIS LESEN
510 PRINT: PRINT PEEK (805)
```

Unter Adresse 805 sollte das Maschinenprogramm den Wert 19 dez abgelegt haben.

Nach Ablauf des Programms finden wir folgendes auf dem Bildschirm:

```
169
19
141
37
3
96

19

]■
```

durch PEEK-Befehle in Zeilen 310 – 330

durch PEEK-Befehl in Zeile 510: richtiges Ergebnis!

Beim Ablauf des BASIC-Programms erfolgt zunächst die Eintragung des Maschinenprogramms in die Speicheradressen 768 . . . 773. Nach Ablage des Maschinenprogramms im Speicher und Wiedergabe der abgespeicherten Codes auf dem Bildschirm läßt Programmzeile 410 das Maschinenprogramm ablaufen. Nach Rückkehr aus dem Maschinenprogramm sieht das BASIC-Programm mit Zeile 510 unter der Speicheradresse 805 dez nach, ob dort der Zahlenwert 19 hinterlegt wurde.

Wir haben mit diesem Beispiel den Brückenschlag von BASIC-Programmen zu MASCHINENprogrammen geschafft und dabei gesehen, wie

- Maschinenprogramme mit POKE-Befehlen im Speicher abgelegt werden können
- Ergebnisse von Maschinenprogrammläufen mit PEEK-Befehlen in ein BASIC-Programm zurückgeholt werden können.

Die Programmierung in Maschinensprache werden wir anhand von Beispielen in den nächsten Kapiteln Schritt für Schritt kennenlernen.

Für die bequeme Eingabe aller in diesem Buch besprochenen Maschinenprogramme schreiben wir uns jedoch zuerst ein einfaches BASIC-Programm – das BBS.

Das BBS

Mit den eben angewandten BASIC-Befehlen PEEK, POKE, CALL werden wir

jetzt ein einfaches BASIC-Programm schreiben, das uns beim Arbeiten mit Maschinenprogrammen dient und folgende Fähigkeiten hat:

- die Hexadezimalcodes eines Maschinenprogramms (Befehle, Daten) in Dialogform vom Benutzer zu erfragen
- dem Benutzer die Korrektur seines eingegebenen Maschinenprogramms zu ermöglichen
- das Maschinenprogramm zu starten
- wenn nötig, Ergebnisse des Maschinenprogrammlaufs auf dem Bildschirm darzustellen
- und Programmerweiterungen für Dialoge zwischen Benutzer und Maschinenprogramm zu ermöglichen.

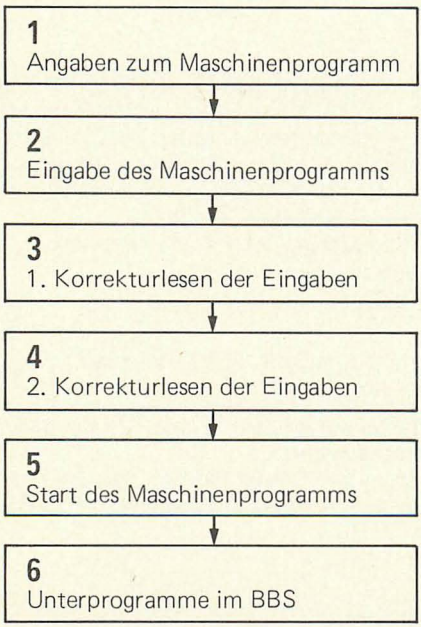
Da Befehle und Daten von Maschinenprogrammen hexadezimal angegeben werden, BASIC-Befehle wie POKE, PEEK, CALL aber nur mit Dezimalzahlen arbeiten, haben wir die Umwandlung hexadezimaler Eingaben in Dezimalzahlen dem BASIC-Programm übertragen.

Da dieses Programm den Charakter eines Betriebssystems hat, haben wir es BASIC-BETRIEBSSYSTEM oder BBS genannt und auf Diskette gespeichert.

Beim Abspeichern des Betriebssystems BBS, etwa mit dem Befehl SAVE BBS, wird nur das BASIC-Programm auf die Diskette übertragen – das mit dem BASIC-Programm eingegebene Maschinenprogramm aber nicht!

Am Ende dieses Buchs beschreiben wir kurz, wie mit dem Betriebssystem BBS eingegebene Maschinenprogramme auf Diskette gespeichert und von anderen BASIC-Programmen wieder geladen und zum Ablauf gebracht werden können.

Das Betriebssystem BBS besteht aus folgenden sechs Programmabschnitten, die wir jetzt im einzelnen besprechen:



Hier die BASIC-Befehle mit Erläuterung zu den einzelnen Programmabschnitten.

Programmabschnitt 1
Angaben zum Maschinenprogramm

Dieser Programmabschnitt erfragt im Dialog mit dem Benutzer folgende Angaben zum Maschinenprogramm in Form von Dezimalzahlen:

- Anfangsadresse des Maschinenprogramms
- Anzahl der Bytes im Maschinenprogramm

Hier das Listing:

```
100  REM *      ANGABEN ZUM     *
101  REM * MASCHINENPROGRAMM   *
102  REM **********************
105  HOME
110  PRINT "ANGABEN ZUM MASCHINEN
     PROGRAMM:"
115  PRINT
120  INPUT "ANFANGSADRESSE  = ";S

125  PRINT
130  INPUT "ANZAHL DER BYTES= ";B

135  PRINT
140  PRINT "MASCHINENPROGRAMM EIN
     GEBEN: "
150 A = S
```

Zeile 105 löscht den Bildschirm. Zeile 120 erfragt die Anfangsadresse und weist sie der Variablen S zu. Zeile 130 erfragt die Anzahl der Bytes und weist sie der Variablen B zu. Zeile 140 fordert zur Eingabe der Codes des Maschinenprogramms auf. Ehe die Eingabe beginnt, wird in Zeile 150 die Anfangsadresse auch in die Variable A geschrieben: dort wird sie im folgenden Programmabschnitt stetig inkrementiert, um die einzelnen Speicheradressen des Maschinenprogramms zu bilden und auf dem Bildschirm darzustellen. In der Variablen S bleibt die Anfangsadresse unverändert erhalten.

Der Dialog mit dem BBS beginnt wie folgt:

```
ANGABEN ZUM MASCHINENPROGRAMM:
ANFANGSADRESSE = 768   <-┐ Ihre Eingaben, jeweils
ANZAHL DER BYTES = 15  <-┘ mit RETURN beendet
MASCHINENPROGRAMM EINGEBEN:
768  □
```

Programmabschnitt 2
Eingabe des Maschinenprogramms

Dieser Programmabschnitt mit den Zeilen 200 . . . 300 legt das eingegebene Maschinenprogramm im Speicher ab. Hierbei übernimmt er folgende Aufgaben:

- Entgegennehmen der Maschinencodes im Dialog mit dem Benutzer
- Prüfen der Eingaben auf Übereinstimmung mit den Hexadezimalzahlen 0 . . . 9, A . . . F
- Umwandeln der hexadezimalen Eingaben in dezimale Form
- Ablegen der eingegebenen Maschinencodes im Speicher
- Darstellen der nächstfolgenden Speicheradresse auf dem Bildschirm

Hier die Befehle dieses Programmabschnitts:

```
200  REM *   HEX EINGEBEN   *
201  REM * HEX/DEZ WANDELN *
202  REM ********************
210  FOR E = 1 TO B
220  PRINT : PRINT A; SPC( 2);
230  GET H$: PRINT H$;
240  GET U$: PRINT U$
250  IF  ASC (H$) < 48 OR  ASC (H
     $) > 70 OR ( ASC (H$) > 57 AND
      ASC (H$) < 65) THEN  PRINT
     "1.ZEICHEN NICHT HEX - ERNEU
     TE EINGABE: ": GOTO 220
260  IF  ASC (U$) < 48 OR  ASC (U
     $) > 70 OR ( ASC (U$) > 57 AND
      ASC (U$) < 65) THEN  PRINT
     "2.ZEICHEN NICHT HEX - ERNEU
     TE EINGABE: ": GOTO 220
270  GOSUB 1000
280  POKE A,D
290 A = A + 1
300  NEXT E
```

Zeilen 210 . . . 300 bilden eine Programmschleife, die entsprechend der Länge B des Maschinenprogramms B-mal durchlaufen wird. Zeile 220 stellt fortlaufend mit den Eingaben am linken Bildschirmrand die Speicheradressen des Maschinenprogramms dar. Mit SPC (2) wird der Cursor in Vorbereitung der Eingabe von der Darstellung der Speicheradresse abgerückt. Zeilen 230, 240 lesen die beiden Hexadezimaleingaben ein und stellen sie mit PRINT auf dem Bildschirm dar. Zeilen 250, 260 prüfen die Eingaben auf nicht-hexadezimale Zeichen und fordern gegebenenfalls mit einer Fehlermeldung eine erneute Eingabe an. In Zeile 270 erfolgt der Sprung in ein Unterprogramm zur Hexadezimal–Dezimalwandlung der Eingaben. Die dezimale

Entsprechung D der beiden hexadezimalen Eingaben H$, U$ wird anschließend in Zeile 280 unter der laufenden Speicheradresse A abgelegt. Bei der Ausführung von POKE wird die Dezimalzahl D in ein entsprechendes Binärmuster umgewandelt. In Zeile 290 wird bei jedem Schleifendurchlauf die Adresse A inkrementiert.

Programmabschnitt 3
Erstes Korrekturlesen der Eingaben

Dieser Programmabschnitt in den Zeilen 400 . . . 530 ermöglicht die Korrektur des eingegebenen Maschinenprogramms und besitzt daher folgende Funktionen:

- Wiedergeben des Maschinenprogramms in Blöcken von 20 Bildschirmzeilen. Adressen werden dezimal, Speicherinhalte hexadezimal dargestellt.
- Entgegennehmen einer neuen Eingabe
- Prüfen der neuen Eingabe
- Umwandlen der Eingabe hexadezimal–dezimal
- Ablegen des korrigierten Maschinencodes unter der zugehörigen Adresse.

Hier die Befehle dieses Programmabschnitts:

```
400  REM * 1.KORREKTURLESEN *
401  REM ********************
410  GOSUB 2000
420  HOME : PRINT "KEINE KORREKTU
     R:  99     EINGEBEN"
430  PRINT : PRINT "FUER  KORREKT
     UR: ADRESSE EINGEBEN"
440  PRINT : INPUT AD
450  IF AD = 99 GOTO 700
460  HOME : PRINT AD; SPC( 3)
470  PRINT "NEUE EINGABE="; SPC(
     2)
480  GET H$: PRINT H$;
490  GET U$: PRINT U$
500  IF  ASC (H$) < 48 OR  ASC (H
     $) > 70 OR ( ASC (H$) > 57 AND
      ASC (H$) < 65) THEN  PRINT
     "1.ZEICHEN NICHT HEX - ERNEU
     TE EINGABE": GOTO 460
510  IF  ASC (U$) < 48 OR  ASC (U
     $) > 70 OR ( ASC (U$) > 57 AND
      ASC (U$) < 65) THEN  PRINT
     "2.ZEICHEN NICHT HEX - ERNEU
     TE EINGABE": GOTO 460
520  GOSUB 1000
530  POKE AD,D
```

In Zeile 410 erfolgt der Sprung in ein Unterprogramm, das jeweils 20 Bytes des Maschinenprogramms in Form von 20 Bildschirmzeilen darstellt. Durch Eingabe von 'RETURN' wird die Darstellung fortgesetzt. In Zeile 440 wird die eingegebene Adresse des zu korrigierenden Maschinencodes der Variablen AD zugewiesen. In Zeile 460 wird zunächst der Bildschirm gelöscht und dann die Adresse AD des zu korrigierenden Maschinencodes dargestellt. In Zeilen 480, 490 wird die Korrektur eingelesen und gleichzeitig auf dem Bildschirm wiedergegeben.

Die Wiedergabe des Maschinenprogramms auf dem Bildschirm endet mit der Frage, ob eine Korrektur erwünscht ist:

```
KEINE KORREKTUR: 99 EINGEBEN
FUER KORREKTUR: ADRESSE EINGEBEN
? □
```

Wird eine Korrektur gewünscht, dann erfragt das Programm die Adresse der zu korrigierenden Eingabe:

```
? 768 □      ← Ihre Eingabe
               abschließen mit RETURN
```

und hierzu die neue Eingabe:

```
768  NEUE EINGABE = □
```

Programmabschnitt 4
Zweites Korrekturlesen

Dieser Programmabschnitt in den Zeilen 600 . . . 640 ermöglicht die Eingabe von mehr als einer Korrektur und stellt schließlich das korrigierte Maschinenprogramm auf dem Bildschirm dar.

Hier die Befehle dieses Programmabschnitts:

```
600  REM * 2.KORREKTURLESEN *
601  REM **********************
605  PRINT
610  PRINT "WEITERE KORREKTUR? J/
     N "
620  GET C$: IF C$ <  > "J" AND C
     $ <  > "N" THEN  GOTO 610
630  IF C$ = "J" THEN  GOTO 430
640  IF C$ = "N" THEN  GOSUB 2000
```

In Zeile 620 wird die Antwort des Benutzers auf die Frage "WEITERE KORREKTUR? J/N" mit GET C$ eingelesen. War die Antwort ein JA, dann erfolgt der Rücksprung in Programmabschnitt 3 (Zeile 630). Andernfalls erfolgt der Sprung in

das "Unterprogramm Programmdarstellung" (Zeile 640).
Hier der Dialog dieses Programmabschnitts mit dem Benutzer:

```
768 NEUE EINGABE = A9  ← Ihre Eingabe
WEITERE KORREKTUR ? J/N  ← Unmittelbar folgende Frage
```

Programmabschnitt 5
Starten des Maschinenprogramms

Dieser Programmabschnitt zwischen den Zeilen 700 . . . 900 löst den Ablauf des Maschinenprogramms aus:

```
700   REM * PROGRAMM STARTEN *
        .
        .
        .

800   CALL S
        .
        .
        .

900   END
```

In den späteren Kapiteln werden Sie zwischen den Zeilen 700 . . . 800 und 800 . . . 900 BASIC-Befehle finden, die beispielsweise dem wiederholten Ablauf eines Maschinenprogramms, dem Rücksprung in den Programmabschnitt "Korrekturlesen" oder Eingaben in das Maschinenprogramm dienen. Zeile 800 enthält den Sprungbefehl CALL zur Anfangsadresse S des Maschinenprogramms. Diese Anfangsadresse S war Ihre erste Eingabe zum Maschinenprogramm. Zeile 900 beendet das BASIC-Programm BBS. Hat das BASIC-Programm den END-Befehl ausgeführt, dann ist das eingegebene Maschinenprogramm nicht mehr über das Betriebssystem BBS in seiner vorliegenden Form aufrufbar. Das Maschinenprogramm steht aber noch ablaufbereit im Speicher und kann entweder durch direkte Eingabe des Befehls CALL 768 oder über ein anderes BASIC-Programm, das den Befehl CALL 768 verwendet, gestartet werden.

Programmabschnitt 6
Unterprogramme im Betriebssystem BBS

Im Betriebssystem BBS treten drei Unterprogramme mit den Funktionen

- Hexadezimal—Dezimal-Wandlung
- Programmdarstellung
- Dezimal—ASCII-Wandlung

auf.

Unterprogramm "Hexadezimal–Dezimal-Wandlung"

Dieses Unterprogramm wandelt die hexadezimal eingegebenen Codes H$, U$ der Maschinenbefehle und Daten in eine Dezimalzahl D um, die vom POKE-Befehl im Hauptprogramm abgespeichert werden kann.

Zu H$, U$ werden nach folgender Tabelle zunächst die ASCII-Codes gefunden, die zwei zusammenhängende Folgen 48 . . . 57 und 65 . . . 70 von Dezimalzahlen bilden. Hieraus werden die Dezimalzahlen 0 . . . 15 durch Subtraktionen M–48, N–48 bzw. M–55, N–55 gewonnen.

H$, U$	M, N	M, N–48 M, N–55
hex	ASCII	dez
0	48	0
1	49	1
2	50	2
3	51	3
4	52	4
5	53	5
6	54	6
7	55	7
8	56	8
9	57	9
A	65	10
B	66	11
C	67	12
D	68	13
E	69	14
F	70	15

Die Dezimalzahl D wird wie folgt aus den beiden Stellen des hexadezimalen Codes gewonnen:

Gewicht der Stellen: 16^1 16^0

Stellen M, N des Hexadezimalcodes: M N

rückgemeldete Dezimalzahl D: $16 \times M + 1 \times N \hat{=} D$

Beispiel: Hex-Code: C 7

ASCII: 67 55

Zeile 1020: neues M = 67 – 55 = 12

Zeile 1050: neues N = 55 – 48 = 7

Zeile 1060: D = 16 x 12 + 1 x 7 = 199

d.h. Wandlung C7 hex → 199 dez

Hier die Befehle des ersten Unterprogramms:

```
1000  REM *  UNTERPROGRAMM    *
1001  REM * HEX/DEZ WANDLUNG *
1002  REM **********************
1010 M =  ASC (H$):N =  ASC (U$)
1020  IF M > 57 THEN M = M - 55: GOTO
      1040
1030 M = M - 48
1040  IF N > 57 THEN N = N - 55: GOTO
      1060
1050 N = N - 48
1060 D = 16 * M + N
1070  RETURN
```

Zeile 1010 bildet die ASCII-Codes M, N zu den eingegebenen Hexadezimalzeichen H$, U$. In Zeilen 1020 . . . 1050 werden die beiden ASCII-Codes M, N als entsprechende Dezimalzahlen ausgedrückt. Zeile 1060 vereinigt die beiden Dezimalzahlen M, N stellenrichtig zur Dezimalzahl D.

Unterprogramm "Programmdarstellung"

Dieses Unterprogramm stellt auf jeweils 20 Bildschirmzeilen den Inhalt von 20 Speicheradressen des Maschinenprogramms dar.

Hier die Befehle des zweiten Unterprogramms:

```
2000  REM *     UNTERPROGRAMM   *
2001  REM * PROGRAMMDARSTELLG *
2002  REM **********************
2010  HOME
2020  PRINT
2030 I = 0:J = 19
2040  ON  INT ((B - 1) / 20) + 1 GOTO
      2090,2080,2070,2060,2050
2050  GOSUB 2200
2060  GOSUB 2200
2070  GOSUB 2200
2080  GOSUB 2200
2090 J = B - 1: GOSUB 2200
2100  RETURN
```

Fortsetzung ⟶

```
2200  HOME : PRINT "HIER IHRE EIN
      GABEN:"
2201  PRINT
2210  FOR E = I TO J
2220  PRINT S + E; SPC( 2);: GOSUB
      3000
2230  NEXT E
2240  PRINT : INPUT "FORTSETZUNG
      MIT 'RETURN'";A$
2250 I = J + 1:J = J + 20
2260  RETURN
```

In Zeile 2030 werden Anfangs- und Endzeiger I, J für die ersten 20 Speicheradressen bzw. Bildschirmzeilen gesetzt. Siehe die Programmschleife in Zeilen 2210 . . . 2230. Zeile 2040 bestimmt aus der angegebenen "Anzahl der Bytes B" die erforderliche Anzahl von 20-Zeilen-Darstellungen. Vorgesehen ist die Wiedergabe von Maschinenprogrammen mit einer Länge von bis zu 100 Bytes. Abhängig von den 5 Fällen: Bis 20, bis 40, bis 60, bis 80, bis 100 Bytes erfolgen GOTO-Sprünge zu 5 Programmzeilen. Jede dieser Programmzeilen 2050 . . . 2090 springt mit GOSUB 2200 in ein Unterprogramm zur Darstellung von jeweils 20 Speicherinhalten. Bei einem Maschinenprogramm bis 20 Bytes erfolgt der Sprung GOTO zu Zeile 2090: Hierdurch wird das Unterprogramm GOSUB 2200 nur einmal aufgerufen; zuvor wird der Endzeiger J auf die tatsächliche Anzahl der Bytes B gesetzt. Bei einem Maschinenprogramm bis 100 Bytes erfolgt der GOTO-Sprung nach Zeile 2050: Von hier wird der Sprung GOSUB 2200 in das darstellende Unterprogramm fünfmal ausgeführt (in Zeilen 2050, 2060, 2070, 2080, 2090).
Zeilen 2210 . . . 2230 stellen eine Programmschleife dar, die fortlaufend mit PRINT S+E die Speicheradressen dezimal auf dem Bildschirm darstellt und den zugehörigen Speicherinhalt durch den Sprung GOSUB 3000 in ein weiteres Unterprogramm bereitstellt. In Zeile 2240 wird die Fortsetzung der jeweils 20zeiligen Bildschirmdarstellungen von einer Eingabe des Benutzers abhängig gemacht.

Unterprogramm "Dezimal–ASCII-Wandlung"

Dieses Unterprogramm liest das im Speicher abgelegte Maschinenprogramm. Der verwendete BASIC-Befehl PEEK meldet die abgelegten Codes des Maschinenprogramms nicht hexadezimal, sondern dezimal zurück. Das vorliegende Unterprogramm führt eine dezimal–hexadezimal-Wandlung aus, so daß die Speicherinhalte hexadezimal dargestellt werden können.

Hier die Befehle des dritten Unterprogramms:

```
3000  REM *   UNTERPROGRAMM     *
3001  REM * DEZ/ASCII WANDLUNG *
3002  REM **********************
3010 Y =  PEEK (S + E)
3020 H =  INT (Y / 16)
3030 U = Y - 16 * H
3040  IF H < 10 THEN  PRINT H;: GOTO
      3060
3050  PRINT  CHR$ (H + 55);
3060  IF U < 10 THEN  PRINT U: GOTO
      3080
3070  PRINT  CHR$ (U + 55)
3080  RETURN
```

In Zeile 3010 wird der Speicherinhalt der augenblicklich dargestellten Speicheradresse S+E gelesen und als Dezimalzahl in Y gespeichert. In Zeilen 3020, 3030 wird die Dezimalzahl Y (deren Wert im Bereich 0 . . . 255 liegt) in hexadezimale Schreibung überführt. In Zeilen 3040, 3050 bzw. 3060, 3070 werden die Hexadezimalzeichen H, U unmittelbar dargestellt, wenn sie eine der Zahlen 0 . . . 9 darstellen. Für 10 . . . 15 erfolgt eine Umwandlung in die Zeichen A . . . F entsprechend der oben mitgeteilten Tabelle.

Vollständiges Listing des Betriebssystems BBS

Hier das vollständige Programm BBS, das Sie als ein ständig benutztes Werkzeug für Ihre Arbeiten mit Maschinenprogrammen in den APPLE-Computer eingeben und etwa mit dem Befehl SAVE BBS auf einer Diskette abspeicher sollten.

```
100  REM *      ANGABEN ZUM     *
101  REM * MASCHINENPROGRAMM   *
102  REM **********************
105  HOME
110  PRINT "ANGABEN ZUM MASCHINEN
     PROGRAMM:"
115  PRINT
120  INPUT "ANFANGSADRESSE  = ";S

125  PRINT
130  INPUT "ANZAHL DER BYTES= ";B
```

Fortsetzung ⟶

```
135  PRINT
140  PRINT "MASCHINENPROGRAMM EIN
     GEBEN: "
150 A = S
200  REM *   HEX EINGEBEN   *
201  REM * HEX/DEZ WANDELN *
202  REM ********************
210  FOR E = 1 TO B
220  PRINT : PRINT A; SPC( 2);
230  GET H$: PRINT H$;
240  GET U$: PRINT U$
250  IF  ASC (H$) < 48 OR  ASC (H
     $) > 70 OR ( ASC (H$) > 57 AND
      ASC (H$) < 65) THEN  PRINT
     "1.ZEICHEN NICHT HEX - ERNEU
     TE EINGABE: ": GOTO 220
260  IF  ASC (U$) < 48 OR  ASC (U
     $) > 70 OR ( ASC (U$) > 57 AND
      ASC (U$) < 65) THEN  PRINT
     "2.ZEICHEN NICHT HEX - ERNEU
     TE EINGABE: ": GOTO 220
270  GOSUB 1000
280  POKE A,D
290 A = A + 1
300  NEXT E
400  REM * 1.KORREKTURLESEN *
401  REM ********************
410  GOSUB 2000
420  HOME : PRINT "KEINE KORREKTU
     R:  99     EINGEBEN"
430  PRINT : PRINT "FUER  KORREKT
     UR: ADRESSE EINGEBEN"
440  PRINT : INPUT AD
450  IF AD = 99 GOTO 700
460  HOME : PRINT AD; SPC( 3)
470  PRINT "NEUE EINGABE="; SPC(
     2)
480  GET H$: PRINT H$;
490  GET U$: PRINT U$
500  IF  ASC (H$) < 48 OR  ASC (H
     $) > 70 OR ( ASC (H$) > 57 AND
      ASC (H$) < 65) THEN  PRINT
     "1.ZEICHEN NICHT HEX - ERNEU
```

```
     TE EINGABE": GOTO 460
510  IF  ASC (U$) < 48 OR  ASC (U
     $) > 70 OR ( ASC (U$) > 57 AND
      ASC (U$) < 65) THEN  PRINT
     "2.ZEICHEN NICHT HEX - ERNEU
     TE EINGABE": GOTO 460
520  GOSUB 1000
530  POKE AD,D
600  REM * 2.KORREKTURLESEN *
601  REM ********************
605  PRINT
610  PRINT "WEITERE KORREKTUR? J/
     N "
620  GET C$: IF C$ <  > "J" AND C
     $ <  > "N" THEN  GOTO 610
630  IF C$ = "J" THEN  GOTO 430
640  IF C$ = "N" THEN  GOSUB 2000

700  REM * PROGRAMM STARTEN *
701  REM ********************
800  CALL S
900  END
1000  REM *  UNTERPROGRAMM   *
1001  REM * HEX/DEZ WANDLUNG *
1002  REM ********************
1010 M =  ASC (H$):N =  ASC (U$)
1020  IF M > 57 THEN M = M - 55: GOTO
     1040
1030 M = M - 48
1040  IF N > 57 THEN N = N - 55: GOTO
     1060
1050 N = N - 48
1060 D = 16 * M + N
1070  RETURN
2000  REM *    UNTERPROGRAMM  *
2001  REM * PROGRAMMDARSTELLG *
2002  REM ********************
2010  HOME
2020  PRINT
2030 I = 0:J = 19
2040  ON  INT ((B - 1) / 20) + 1 GOTO
     2090,2080,2070,2060,2050
2050  GOSUB 2200
```

Fortsetzung →

```
2060  GOSUB 2200
2070  GOSUB 2200
2080  GOSUB 2200
2090 J = B - 1: GOSUB 2200
2100  RETURN
2200  HOME : PRINT "HIER IHRE EIN
      GABEN:"
2201  PRINT
2210  FOR E = I TO J
2220  PRINT S + E; SPC( 2);: GOSUB
      3000
2230  NEXT E
2240  PRINT : INPUT "FORTSETZUNG
      MIT 'RETURN'";A$
2250 I = J + 1:J = J + 20
2260  RETURN
3000  REM *   UNTERPROGRAMM     *
3001  REM * DEZ/ASCII WANDLUNG *
3002  REM **********************
3010 Y =  PEEK (S + E)
3020 H =  INT (Y / 16)
3030 U = Y - 16 * H
3040  IF H < 10 THEN  PRINT H;: GOTO
      3060
3050  PRINT  CHR$ (H + 55);
3060  IF U < 10 THEN  PRINT U: GOTO
      3080
3070  PRINT  CHR$ (U + 55)
3080  RETURN
```

6
GRAPHIK - und BBS

Einführung

Wir werden in diesem Kapitel Maschinenprogramme für graphische Darstellungen kennenlernen – und diese Maschinenprogramme über unser Betriebssystem BBS in den APPLE eingeben.

Für die Elemente Punkt und Linie in graphischen Darstellungen und für Ansteuerungen des Bildschirms verfügt der APPLE-Computer über fest eingespeicherte Maschinenprogramme:

Fest eingespeicherte Maschinenprogramme im APPLE

Anfangsadresse im Speicher dez	hex	Funktion
63 488	F8ØØ	PUNKT DARSTELLEN
63 513	F819	HORIZONTALE LINIE DARSTELLEN
63 528	F828	VERTIKALE LINIE DARSTELLEN
64 320	FB4Ø	GRAPHIK MODE SETZEN
64 600	FC58	BILDSCHIRM LÖSCHEN
64 795	FD1B	ZUFALLSZAHL ZIEHEN
64 821	FD35	TASTENEINGABE LESEN
64 910	FD8E	SPRUNG AUF FOLGENDEN ZEILENANFANG
64 986	FDDA	AKKUMULATORINHALT HEX. DARSTELLEN
65 517	FDED	AKKUMULATORINHALT ASCII DARSTELLEN
65 338	FF3A	SIGNALTON DES LAUTSPRECHERS

Wir werden in diesem Kapitel erfahren, wie wir diese APPLE-Maschinenprogramme für uns einsetzen können. Dadurch ersparen wir uns sehr viel Arbeit, da wir diese APPLE-Routinen auch in künftigen Programmen immer wiederverwenden können.

Einen Punkt darstellen

In diesem Programm machen wir von drei der fest eingespeicherten Maschinenprogramme im APPLE-Computer Gebrauch. Wie bei einem Programm in BASIC machen wir uns einen Plan der einzelnen Programmschritte:

1. BILDSCHIRM LÖSCHEN
2. IN GRAPHIKMODE GEHEN
3. FARBE WÄHLEN
4. PUNKTKOORDINATE WÄHLEN
5. PUNKT DARSTELLEN
6. RÜCKKEHR ZUM BBS

Für die Eingabe des Maschinenprogramms über das Betriebssystem BBS benötigen wir folgende ANGABEN ZUM MASCHINENPROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 18

Hier die Maschinencodes, die das BBS bei Eingabe des **Maschinenprogramms "Einen Punkt darstellen"** von uns erfragt:

1. Bildschirm löschen

768 20 ← JSR FC58 769 58 770 FC	Sprung in das Maschinenprogramm "BILDSCHIRM LÖSCHEN" unter Anfangsadresse FC58

2. Graphikmode herstellen

771 20 ← JSR FB40 772 40 773 FB	Sprung in das Maschinenprogramm "GRAPHIKMODE" unter Anfangsadresse FB40

3. Farbe bestimmen

774 A9 ← LDA FF 775 FF	Farbwert FF in Akkumulator laden
776 85 ← STA 0030 777 30	Farbwert FF unter Adresse 0030 abspeichern

4. Koordinaten laden

778 A0 ← LDY 05 779 05	Spalte 05 hex in 4-Register laden
780 A9 ← LDA 20 781 20	Zeile 20 hex in Akkumulator laden

5. Punkt darstellen

782 20 ← JSR F800 783 00 784 F8	Sprung in das Maschinenprogramm "PUNKT DARSTELLEN" unter Anfangsadresse F800

6. Rücksprung

785 60 ← RTS

Wir gehen jetzt Schritt für Schritt durch dieses Maschinenprogramm.

SCHRITT 1

Adressen 768, 769, 770. Das Programm beginnt mit dem Sprung in das festeingespeicherte Maschinenprogramm BILDSCHIRM LÖSCHEN im APPLE-Computer. Der Sprungbefehl JSR (JUMP TO SUBROUTINE) wird hexadezimal durch den Code 20 dargestellt. Die Anfangsadresse des Maschinenprogramms im APPLE-Computer ist FC58 hex (oder 64600 dez). Beachten Sie die Reihenfolge der Adreßeingabe: Zuerst das LSB 58, dann das MSB FC.

SCHRITT 2

Adressen 771, 772, 773. In diesem Programmschritt wird das Darstellungsformat auf

dem Bildschirm geändert. Im Graphikmode zerfällt der Bildschirm in einen oberen Darstellungsbereich mit 40 x 40 Bildpunkten und einem unteren Darstellungsbereich von 4 Textzeilen. Diese Änderung der Darstellungsform übernimmt im APPLE-Computer ein fest eingespeichertes Maschinenprogramm, das im Speicher mit der Adresse FB40 hex beginnt. Mit dem Code 20 hex in Adresse 771 geben wir den Sprungbefehl JSR ein; mit den Codes 40 hex und FB hex in Adressen 772, 773 teilen wir die Adresse FB40 zu der der Sprung erfolgen soll.

SCHRITT 3

In diesem Programmschritt können wir die Farbe des darzustellenden Punkts festlegen. Hier die Farbtabelle, aus der wir auswählen können:

Farbcodes im niedrigauflösenden Graphikmode:

Farbcode		Farbe
dez	hex	
0	0	Schwarz
1	1	Magenta
2	2	Dunkelblau
3	3	Hellpurpur
4	4	Dunkelgrün
5	5	Grau
6	6	Mittelblau
7	7	Hellblau
8	8	Braun
9	9	Orange
10	A	Grau
11	B	Pink
12	C	Grün
13	D	Gelb
14	E	Blau/Grün
15	F	Weiß

Der gewählte Farbwert muß als Hexadezimalcode im Speicherplatz mit der Adresse **0030 hex** abgelegt werden, wo ihn ein später aufgerufenes Maschinenprogramm zur Darstellung des Punkts erwartet.

Wir wählen die Farbe Weiß mit dem Code F. Dieser Wert muß zunächst in den Akkumulator A des Mikroprozessors geladen und von dort mit einem Speicherbefehl zur Adresse 0030 hex gebracht werden. Diese Aufgaben lösen die Maschinenbefehle LDA (LOAD ACCUMULATOR) und STA (STORE ACCUMULATOR IN MEMORY).

Adressen 774, 775. Der Code A9 hex bezeichnet eine Variante des Ladebefehls LDA, bei der der Akkumulator A nicht mit dem Inhalt eines adressierten Speicherplatzes, sondern mit demjenigen Wert geladen wird, der unmittelbar auf dem LDA-Befehl folgt (siehe Anhang D). Der Farbwert F hex wird in beide Bytes der Adresse 775 geschrieben.

Adressen 776, 777. Mit dem Code 85 hex wird eine Version des Speicherbefehls STA ausgewählt, mit der der Inhalt des Akkumulators A in den Speicherbereich 0000 . . .

00FF hex übertragen werden kann. Dieser Speicherbereich, auch Page Zero oder Nullseite genannt, benötigt als Adreßangabe 1 Byte. (Das höchstwertige Byte MSB ist in diesem Speicherbereich immer 00). In Adresse 777 steht daher der Code 30 hex für den Speicherplatz mit der Adresse 0030 hex, in dem der Farbwert nach der oben mitgeteilten Tabelle stehen muß.

Will man den Farbwert ändern, dann gibt man mit den Korrekturmöglichkeiten des Betriebsystems BBS in Adresse 775 einen der anderen Tabellenwerte an. Beispielsweise würde für die Farbe Braun in Adresse 775 der Code 88 hex bei der Programmeingabe eingetragen werden.

SCHRITT 4

Dieser Programmabschnitt legt die Koordinaten des darzustellenden Punkts im 40x40-Raster des graphischen Darstellungsfelds auf dem Bildschirm fest. Die Spaltenangabe muß in das Y-Register, die Zeilenangabe in den Akkumulator A geladen werden. Im Beispiel haben wir Spalte 5, Zeile 32 (20 hex) gewählt.

Adressen 778, 779. Mit dem Code A0 hex haben wir eine Version des Befehls LDY zum Laden des Y-Registers gewählt, bei der der zu ladende Wert unmittelbar nach dem Ladebefehl steht, hier also in Adresse 779 (siehe auch Anhang D).

Adressen 780, 781. Mit dem Code A9 hex wird auch für den Akkumulator A eine Version des Ladebefehls gewählt, bei der der zu ladende Wert beim Befehl steht. In diesem Fall die Spaltenangabe 20 hex.

Zur Veränderung der Koordinaten (05, 20) des Punkts können wir mit der Korrekturmöglichkeit des Betriebssystems BBS in die Adressen (779, 781) andere Werte aus dem Bereich 00 . . . 27 hex eintragen.

SCHRITT 5

Dieser Programmabschnitt spricht ein festeingespeichertes Maschinenprogramm im APPLE-Computer an, das einen Punkt im 40x40-Raster des Bildschirms entsprechend den Angaben von Farbe und Koordinaten darstellt.

Adressen 782, 783, 784. Mit dem Code 20 hex wird wieder der Sprungbefehl JSR angegeben. Die Sprungadresse ist diesmal F800 hex, die Anfangsadresse des Maschinenprogramms PUNKT DARSTELLEN im APPLE-Computer.

SCHRITT 6

Jedes Maschinenprogramm, das wir schreiben, muß mit einem RETURN-Befehl abgeschlossen werden, der als Maschinenbefehl die Kurzbezeichnung RTS trägt und mit dem Code 60 hex dargestellt wird.

Bei den zuvor mit JSR FC58, JSR FB40 und JSR F800 aufgerufenen Maschinenprogrammen im APPLE-Computer ist der RTS-Befehl bereits als Schlußbefehl enthalten und muß von uns nicht gesondert eingegeben werden.

Der Maschinenbefehl RTS läßt das aufrufende Betriebsprogramm BBS mit demjenigen Befehl fortsetzen, der auf dem Befehl CALL S in Zeile 800 folgt. Dies ist beispielsweise in Zeile 900 der BASIC-Befehl END zum Beenden des BBS.

Eckpunkte eines Rechtecks darstellen

Im vorigen Maschinenprogramm haben wir mit den Befehlen (LDY, LDA) die (Koordinatenspalte, Zeile) eines einzelnen Punkts eingegeben und durch den Sprung JSR F800 in das Maschinenprogramm PUNKT DARSTELLEN auf den Bildschirm gebracht.

Das vorige Maschinenprogramm wurde um Befehle erweitert, mit denen sich 3 weitere Punkte darstellen lassen. Als Beispiel haben wir die Eckpunkte des graphischen Darstellungsfelds gewählt:

SPALTE		ZEILE		
dez	hex	dez	hex	
Ø	Ø	Ø	Ø	oben links
Ø	Ø	39	27	unten links
39	27	Ø	Ø	oben rechts
39	27	39	27	unten rechts

Die ANGABEN ZUM MASCHINENPROGRAMM lauten jetzt:

Anfangsadresse: 768
Anzahl der Bytes: 35

Hier die hexadezimalen Codes des **Maschinenprogramms "Eckpunkte eines Rechtecks darstellen":**

1. Bildschirm löschen
   768 2Ø JSR FC58
   769 58
   770 FC

2. Graphikmode herstellen
   771 2Ø JSR FB40
   772 4Ø
   773 FB

3. Farbe bestimmen
   774 A9 LDA FF
   775 FF
   776 85 STA 0030
   777 3Ø

4. Spalte 0, Zeile 0 laden
   778 AØ LDY 00 Spalte 0
   779 ØØ
   780 A9 LDA 00 Zeile 0
   781 ØØ

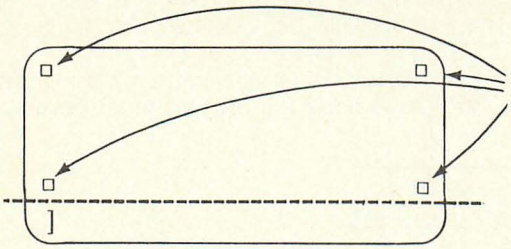
5. Linke, obere Ecke
   782 2Ø JSR F800
   783 ØØ
   784 F8
   785 A9 LDA 27 Zeile 27
   786 27

6. Linke, untere Ecke
   787 2Ø JSR F800
   788 ØØ
   789 F8
   790 AØ LDY 27 Spalte 27
   791 27
   792 A9 LDA 27 Zeile 27
   793 27

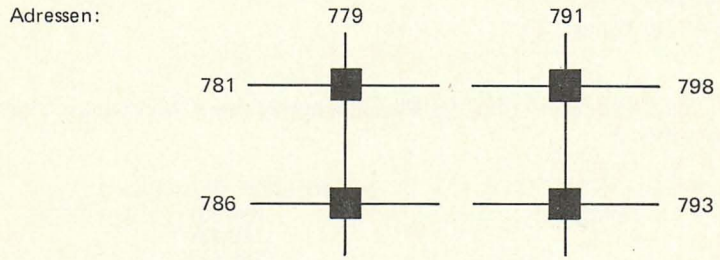
7. Rechte, untere Ecke
   794 2Ø JSR F800
   795 ØØ
   796 F8
   797 A9 LDA 00 Zeile 0
   798 ØØ

8. Rechte, obere Ecke
   799 2Ø JSR F800
   800 ØØ
   801 F8
   802 6Ø RTS

Nach Ablauf des Maschinenprogramms finden wir folgende Darstellung auf dem Bildschirm:



Die Koordinaten der 4 Eckpunkte lassen sich mit der Korrekturmöglichkeit im Betriebssystem BBS leicht verändern. Wir müssen lediglich an folgenden Adressen neue Hexadezimalwerte ablegen:



Zur Änderung der Farbe aller 4 Eckpunkte müssen wir den Hexadezimalwert FF in Adresse 775 entsprechend der Farbtabelle ändern.

Eine horizontale Linie darstellen

Zur Darstellung einer horizontalen Linie auf dem Bildschirm können wir ein fest eingespeichertes Maschinenprogramm mit der Anfangsadresse F819 hex benutzen.

Diesem Maschinenprogramm muß eine horizontale Linie durch folgende Angaben beschrieben werden:

- Farbe
  Der Farbcode nach der Farbtabelle muß in der Speicherstelle mit der Adresse **0030 hex** abgelegt werden.
- Endpunkt
  Das Ende der horizontalen Linie wird durch eine der Spaltennummern 00 . . . 27 hex beschrieben. Diese Spaltennummer muß in der Speicherstelle mit der Adresse **002C hex** abgelegt werden.

- Anfangspunkt
  Der Anfangspunkt wird mit seinen Koordinaten Spalte, Zeile geschrieben. Die Spaltennummer wird in das Y-Register, die Zeilennummer in den Akkumulator A geladen.

In unserem Beispiel stellen wir folgende horizontale Linie dar:

Farbe:	Weiß, Code F hex
Endpunkt:	Spalte 32 (20 hex)
Anfangspunkt:	Spalte 16 (10 hex)
	Zeile 20 (14 hex)

Zur Programmeingabe über das BBS gehören folgende ANGABEN ZUM MASCHINENPROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 22

Hier die einzugebenden Codes des **Maschinenprogramms "Eine horizontale Linie darstellen":**

1. Bildschirm löschen

768 20	JSR FC58	
769 58		
770 FC		

2. Graphikmode herstellen

771 20	JSR FB40	
772 40		
773 FB		

3. Farbe bestimmen

774 A9	LDA FF	Farbwert FF = WEISS laden
775 FF		
776 85	STA 30	Farbwert unter 0030 abspeichern
777 30		

4. Endpunkt angeben

778 A9	LDA 20	Spalte des Endpunkts 20 hex = 32 dez laden
779 20		
780 85	STA 2C	Endpunkt unter 002C abspeichern
781 2C		

5. Anfangspunkt angeben

782 A0	LDY 10	Spalte des Anfangspunkts 10 hex = 10 dez laden
783 10		
784 A9	LDA 14	Zeile des Anfangspunkts 14 hex = 20 dez laden
785 14		

6. Linie darstellen

786 20	JSR F819	Sprung in Maschinenprogramm "PUNKT DARSTELLEN"
787 19		unter Anfangsadresse F819
788 F8		

7. Rücksprung

789 60	RTS	

Hier Erklärungen zu einigen der Programmschritte:

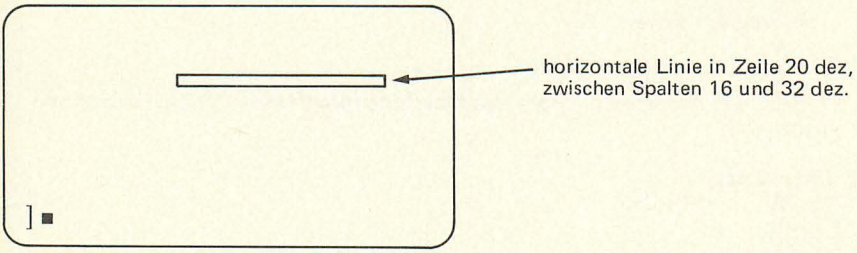
SCHRITT 4

In diesem Programmabschnitt wird die Spaltennummer 20 hex mit dem Befehl LDA 20 zunächst in den Akkumulator und von dort mit dem Befehl STA 2C in die Speicherstelle mit der Adresse 002C hex geladen.

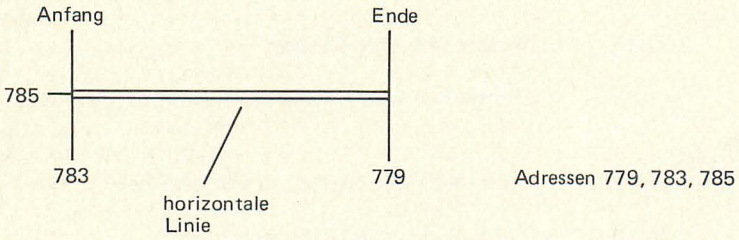
SCHRITT 6

Nach Beschreibung der horizontalen Linie durch 4 Angaben wird mit dem Sprung JSR F819 ein fest im APPLE-Computer eingespeichertes Maschinenprogramm benutzt, das diese Linie auf dem Bildschirm darstellt.

Nach Ablauf des Maschinenprogramms finden Sie auf dem Bildschirm folgende Darstellung:



Für Programmwiederholung mit neuen Liniendarstellungen verändern Sie die Codes unter folgenden Adressen:



Das dargestellte Maschinenprogramm entspricht in seiner Funktion folgendem BASIC-Befehl:

HLIN 16, 32 AT 20

Eine vertikale Linie darstellen

Auch zur Darstellung einer vertikalen Linie können wir ein fest eingespeichertes Maschinenprogramm im APPLE-Computer benutzen. Dieses hat die Anfangsadresse F828 hex.

Im vorigen Maschinenprogramm müssen wir folgende Änderungen ausführen, um vertikale Linien darstellen zu können:

Adresse 781
Statt 2C muß der Wert 2D eingetragen werden.
Der Endpunkt vertikaler Linien muß unter Adresse **002D hex** abgelegt sein.

Adresse 787
Statt 19 muß der Wert 28 eingetragen werden.
Die Anfangsadresse des Maschinenprogramms für vertikale Linien lautet **F828 hex.**

1. Bildschirm löschen

```
768 2Ø      JSR FC58
769 58
770 FC
```

2. Graphikmode herstellen

```
771 2Ø      JSR FB40
772 4Ø
773 FB
```

3. Farbe bestimmen

```
774 A9      LDA FF
775 FF

776 85      STA 30
777 3Ø
```

4. Endpunkt angeben

```
778 A9      LDA 20
779 2Ø

780 85      STA 2D      Änderung: Adresse 002D
781 2D
```

5. Anfangspunkt angeben

```
782 AØ      LDY 14      Änderung: 14 hex = 20 dez
783 14

784 A9      LDA 1Ø      Änderung: 1Ø hex = 16 dez
785 1Ø
```

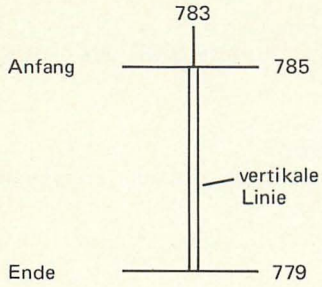
6. Linie darstellen

```
786 2Ø      JSR F828    Änderung: Anfangsadresse F828
787 28
788 F8
```

7. Rücksprung

```
789 6Ø
```

Die übrigen Korodinatenangaben bleiben in ihren Speicherstellen, haben aber jetzt eine andere Bedeutung:



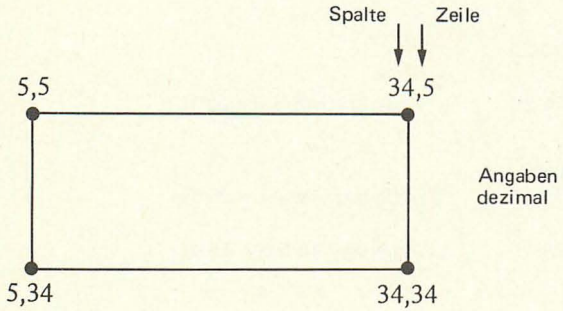
Adressen 779, 783, 785

Die Funktion des Maschinenprogramms für die Darstellung vertikaler Linien auf dem Bildschirm entspricht der des BASIC-Befehls:

VLIN 16, 32 AT 20

Ein Rechteck darstellen

Dieses Programm ist eine Mischung aus den beiden vorangegangenen Programmen. Wir haben als Beispiel ein Rechteck mit folgenden Koordinaten gewählt:



Die Programmeingabe beginnt mit folgenden ANGABEN ZUM MASCHINEN-PROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 45

Hier die einzugebenden Codes für das **Maschinenprogramm "Ein Rechteck darstellen":**

1. Bildschirm löschen

```
768 20      JSR FC58
769 58
770 FC
```

2. Graphikmode herstellen

```
771 20      JSR FB40
772 40
773 FB
```

3. Farbe bestimmen

```
774 A9      LDA FF
775 FF

776 85      STA 0030
777 30
```

4. Endpunkt angeben

```
778 A9      LDA 22      Endpunkt in Spalte/Zeile 22 hex = 34 dez
779 22

780 85      STA 2C      Endspalte unter 002C abspeichern
781 2C

782 85      STA 2D      Endzeile unter 002D abspeichern
783 2D
```

5. Anfangspunkt angeben

```
784 A0      LDY 05      Spalte des Anfangspunkts 5 laden
785 05

786 A9      LDA 05      Zeile des Anfangspunkts 5 laden
787 05
```

6. Obere Seite darstellen

```
788 20      JSR F819
789 19
790 F8
```

7. Neue Anfangswerte

```
791 A0      LDY 05      Unveränderte Anfangsspalte
792 05

793 A9      LDA 22      Neue Zeile 22 hex = 34 dez
794 22
```

8. Untere Seite darstellen

```
795 20      JSR F819
796 19
797 F8
```

9. Neue Anfangswerte

```
798 A0      LDY 05      Spalte des Anfangspunkts 5 laden
799 05

800 A9      LDA 05      Zeile des Anfangspunkts 5 laden
801 05
```

10. Linke Seite darstellen

```
802 20      JSR F828
803 28
804 F8
```

→

11. Neue Anfangswerte

```
805 AØ      LDY 22      Spalte des Anfangspunkts 22 hex = 34 dez laden
806 22
807 A9      LDA 05      Zeile des Anfangspunkts 5 laden
808 Ø5
```

12. Rechte Seite darstellen

```
809 2Ø      JSR F828
810 28
811 F8
```

13. Rücksprung

```
812 6Ø      RTS
```

7
TEXT – und BBS

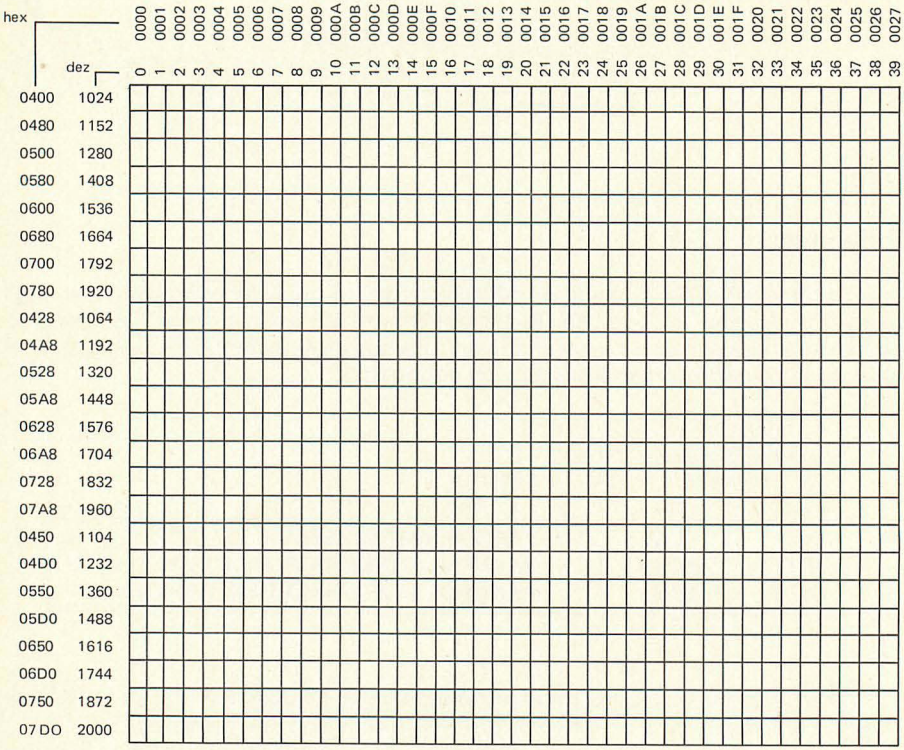
Einführung

In diesem Kapitel erfahren wir Möglichkeiten, mit Maschinenprogrammen Text auf dem Bildschirm darzustellen. In Maschinenprogrammen zur Texteingabe und Textdarstellung können wir vorteilhaft von folgenden fest in den APPLE eingespeicherten Maschinenprogrammen Gebrauch machen:

Anfangsadresse im Speicher		Funktion
dez	hex	
64 821	FD35	TASTENEINGABE LESEN
65 517	FDED	AKKUMULATORINHALT ASCII DARSTELLEN
64 986	FDDA	AKKUMULATORINHALT HEX. DARSTELLEN

Das Darstellungsfeld des Bildschirms besteht für Textdarstellungen aus 40 Spalten und 24 Zeilen. Den 24 x 40 = 960 Zeichenfeldern sind 960 Speicherplätze im Adreßbereich 1024 . . . 2047 zugeordnet. Hier der Zusammenhang zwischen Speicheradressen und Zeichenfeldern des Bildschirms:

Adressen



Legt man unter einer dieser Speicheradressen den ASCII-Code eines Zeichens ab, dann wird dieses Zeichen im zugehörigen Zeichenfeld dargestellt. Folgende Darstellungsformen von Text werden wir kennenlernen:

- normal
- invertiert
- blinkend

Eine vollständige Zusammenstellung der ASCII-Codes für diese Darstellungsart findet sich in Anhang C-3. Hier ein Auszug:

Zeichen	Darstellung		
	nor	bl	inv
@	CØ	4Ø	ØØ
A	C1	41	Ø1
B	C2	42	Ø2
C	C3	43	Ø3
D	C4	44	Ø4
E	C5	45	Ø5
F	C6	46	Ø6
G	C7	47	Ø7
H	C8	48	Ø8
I	C9	49	Ø9
J	CA	4A	ØA
K	CB	4B	ØB
L	CC	4C	ØC
M	CD	4D	ØD
N	CE	4E	ØE
O	CF	4F	ØF
P	DØ	5Ø	1Ø
Q	D1	51	11
R	D2	52	12
S	D3	53	13
T	D4	54	14
U	D5	55	15
V	D6	56	16
W	D7	57	17
X	D8	58	18
Y	D9	59	19
Z	DA	5A	1A

nor normal
bl blinkend
inv invertiert

Wir werden einfache Lösungen kennenlernen, zeilenweise Text auf den Bildschirm schreiben, ohne die Speicheradresse jeder Zeile und jedes Zeichenfelds bestimmen zu müssen.

Ein einzelnes Zeichen darstellen

Wir beginnen mit einem einfachen Programm, das den Buchstaben A in die linke obere Ecke des Bildschirms schreibt.

Das darzustellende Zeichen wird als hexadezimaler ASCII-Code zunächst in den Akkumulator A geladen und von dort in die Speicherstelle mit der Adresse 1024 dez oder 0400 hex übertragen. Die Speicheradresse 1024 entspricht dem ersten Zeichenfeld auf dem Bildschirm. Ist das Zeichen im Bildschirmspeicher abgelegt, wird es auch sofort auf dem Bildschirm dargestellt.

Im Dialog mit dem BBS machen wir folgende ANGABEN ZUM MASCHINENPROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 9

Hier die einzugebenden Codes des **Maschinenprogramms zur Darstellung eines Einzelzeichens:**

1. Bildschirm löschen

768	20	JSR FC58	Sprung in Maschinenprogramm "BILDSCHIRM
769	58		LÖSCHEN" unter Anfangsadresse FC58
770	FC		

2. Zeichen wählen

771	A9	LDA C1	ASCII-Code C1 für 'A' laden
772	C1		

3. Buchstaben darstellen

773	8D	STA 040	ASCII-Code in Bildschirmspeicher unter Adresse
774	00		0400 hex = 1024 dez speichern und damit darstellen
775	04		

4. Rücksprung ins aufrufende Programm

776	60	RTS	

Hier Erläuterungen zu einigen Programmschritten:

SCHRITT 2

Adressen 771, 772. Der Maschinenbefehl LDA C1 mit dem Code A9 hex lädt den ASCII-Code des Buchstaben A, C1, unmittelbar in den Akkumulator A.

SCHRITT 3

Adressen 773, 774, 775. Der Speicherbefehl STA 0400 mit dem Operationscode 8D überträgt den Inhalt des Akkumulators A in den Speicherplatz mit der absoluten Adresse 0400 hex oder 1024 dez. Dies ist die Adresse des ersten Zeichenfelds auf dem Bildschirm. Mit Ausführung des Befehls STA wird das zum ASCII-Code C1 gehörende Zeichen A auf dem Bildschirm dargestellt.

Eine Zeile mit Zeichen füllen

Das folgende Maschinenprogramm füllt die erste Bildschirmzeile mit einem frei wählbaren Zeichen.

Das Maschinenprogramm muß hierfür nacheinander die Adressen 0400 . . . 0427 aller Zeichenfelder in der ersten Bildschirmzeile bilden. Für diese Aufgabe benutzen wir einen Speicherbefehl STA mit indizierter Adressierung.

Hier die ANGABEN ZUM MASCHINENPROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 16

Hier die einzugebenden Codes des **Maschinenprogramms zum Füllen der 1. Bildschirmzeile:**

```
1. Bildschirm löschen
     768 20      JSR FC58
     769 58
     770 FC

2. Schleifenzähler setzen
     771 A0      LDY 0          Zähler (Y-Register) auf Wert 0 setzen
     772 00

3. Zeichen wählen
     773 A9      LDA C1         ASCII-Code für 'A' laden
     774 C1

4. Programmschleife
  ┌─→ 775 99     STA 0400, Y    ASCII-Code in Adressen 0400 + 0, 0400 + 1, . . .
  │  776 00                     speichern
  │  777 04
  │  778 C8     INY            Zähler um Wert 1 erhöhen
  │  779 C0     CPY 28         Zählerinhalt mit Wert 28 hex = 40 dez vergleichen
  │  780 28
  │  781 D0     BNE F8         Verzweigen nach 775, wenn Zählerinhalt ≠28 hex
  └─ 782 F8

5. Rücksprung ins aufrufende Programm
     783 60      RTS
```

Hier Erläuterungen einiger Programmschritte:

SCHRITT 2

Das Y-Register dient im weiteren Verlauf als Adreßzeiger zu den 40 Zeichenfeldern der ersten Bildschirmzeile. Mit LDY 0 wird das Register auf seinen Anfangswert 00 hex gesetzt.

SCHRITT 3

Mit LDA C1 wird der ASCII-Code des Buchstaben A unmittelbar in den Akkumulator A geladen.

SCHRITT 4

Dieser Programmabschnitt schreibt den Inhalt des Akkumulators A in die Adressen 0400 . . . 0427 hex der ersten 40 Zeichenfelder des Bildschirms – und bringt damit den Akkumulatorinhalt zur Darstellung.

In einer Programmschleife wird hierzu der Speicherbefehl STA 40mal ausgeführt, wobei jedesmal die absolute Anfangsadresse 0400 in der Form 0400+Y um den inkrementierten Inhalt des Y-Registers erhöht wird.

Hier die Maschinenbefehle, mit denen die Programmschleife gebildet wurde:

Adressen 775, 776, 777. Mit dem Operationscode 99 hex wird eine Version des STA-

Befehls gewählt, bei der eine absolute Adresse (hier 0400) mit dem Inhalt des Y-Registers indiziert wird.

Adresse 778. Mit C8 hex wird der Inkrementierungsbefehl INY dargestellt, der bei jeder Ausführung den Inhalt des Y-Registers um den Wert 1 erhöht.

Adressen 779, 780. Der Code C0 hex stellt eine Version des Vergleichsbefehl CPY dar, bei der der Inhalt des Y-Registers mit einem unmittelbar nach CPY angegebenen Wert, hier 28 hex oder 40 dez, verglichen wird. Dieser Befehl subtrahiert den Wert 28 vom Inhalt des Y-Registers und setzt das Bit Z im Statusregister des Mikroprozessors auf den Wert 1, falls beide Werte gleich groß waren.

Adressen 781, 782. Der Verzweigungsbefehl BNE mit dem Operationscode D0 hex führt eine Verzweigung dann aus, wenn die Werte des vorangehenden Vergleichs ungleich waren (das Bit Z nicht den Wert 1 hat). Die Verzweigung erfolgt relativ, in diesem Fall um F8 hex oder –8 dez Schritte zurück zur Adresse 775. Zum Auszählen der Sprungweite F8 muß man wissen, daß der Programmzähler im Mikroprozessor während der Ausführung des Befehls BNE bereits die Adresse des nachfolgenden Befehls, hier also 783, geladen hat. Bei einem Rücksprung im Programm ist die gefundene Anzahl von Schritten, um die der Programmzähler zurückgesetzt werden muß, als negative Hexadezimalzahl anzugeben. Hierzu benutzen wir die Tabelle in Anhang B, die auszugsweise hier wiedergegeben wird:

VORVERZWEIGUNGEN

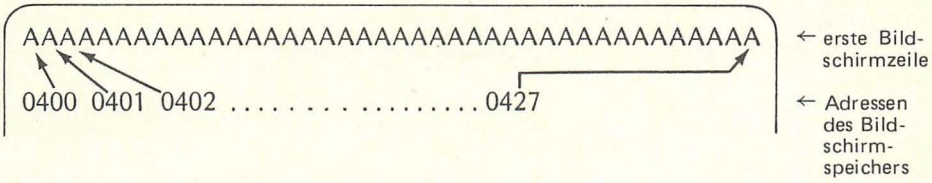
Verzweigungsschritte		Verzweigungsschritte		Verzweigungsschritte	
dez	hex	dez	hex	dez	hex
1	01	49	31	97	61
2	02	50	32	98	62
3	03	51	33	99	63
4	04	52	34	100	64
5	05	53	35	101	65
6	06	54	36	102	66
7	07	55	37	103	67
8	08	56	38	104	68
9	09	57	39	105	69
10	0A	58	3A	106	6A
11	0B	59	3B	107	6B
12	0C	60	3C	108	6C
13	0D	61	3D	109	6D
14	0E	62	3E	110	6E
15	0F	63	3F	111	6F
16	10	64	40	112	70

RÜCKVERZWEIGUNGEN

Verzweigungsschritte		Verzweigungsschritte		Verzweigungsschritte	
dez	hex	dez	hex	dez	hex
–1	FF	–49	CF	–97	9F
–2	FE	–50	CE	–98	9E
–3	FD	–51	CD	–99	9D
–4	FC	–52	CC	–100	9C
–5	FB	–53	CB	–101	9B
–6	FA	–54	CA	–102	9A
–7	F9	–55	C9	–103	99
–8	F8	–56	C8	–104	98
–9	F7	–57	C7	–105	97
–10	F6	–58	C6	–106	96
–11	F5	–59	C5	–107	95
–12	F4	–60	C4	–108	94
–13	F3	–61	C3	–109	93
–14	F2	–62	C2	–110	92
–15	F1	–63	C1	–111	91
–16	F0	–64	C0	–112	90

Die Schleifendurchläufe werden beendet, wenn das Y-Register die Werte 0 . . . 39 durchlaufen hat und schließlich auf den Wert 40 dez inkrementiert wird.

Nach Ablauf des Maschinenprogramms finden wir folgende Darstellung auf dem Bildschirm:



Die Anfangsadresse 0400 hex der ersten Bildschirmzeile können wir auch in die Anfangsadresse jeder anderen Bildschirmzeile verändern. Siehe hierzu die Darstellung am Anfang dieses Kapitels. Die Anfangsadresse der dritten Bildschirmzeile ist beispielsweise 0500.

Das Alphabet darstellen

Im vorigen Maschinenprogramm haben wir erstmals eine indizierte Adressierung angewendet, bei der das Y-Register den Adreßzeiger oder Index bildete.

Im folgenden Programm bilden wir mit den Registern Y und X gleichzeitig zwei Adreßzeiger. Dabei bilden wir wieder mit dem Y-Register alle Adressen der ersten Bildschirmzeile, während wir mit dem X-Register 26 Adressen in einen Speicherbereich bilden, der die 26 Buchstaben A . . . Z in Form ihrer ASCII-Codes gespeichert hält.

An diesem Programm erfahren wir eine Besonderheit der Bildung von Datenlisten in Maschinenprogrammen. Da für den Mikroprozessor Daten und Befehle gleich aussehen – beide stehen als 0-1-Kombinationen im Speicher – müssen Datenlisten aus dem eigentlichen Maschinenprogramm ausquartiert und in Speicherbereichen vor oder nach dem Maschinenprogramm abgelegt werden. In unserem Beispiel beginnt die Datenliste nach dem letzten Maschinenbefehl RTS.

Die Programmeingabe beginnt mit folgenden ANGABEN ZUM MASCHINENPROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 46

Hier die Codes des **Maschinenprogramms zur Darstellung des Alphabets:**

1. Initiierung

768	2Ø	JSR FC58	Bildschirm löschen
769	58		
770	FC		
771	AØ	LDY 0	Y-Register auf Wert 0 setzen
772	ØØ		
773	A2	LDX 0	X-Register auf Wert 0 setzen
774	ØØ		

2. Programmschleife

775	BD	LDA 0314, X	Akkumulator von Speicheradressen 0314 + 0, 0314 + 1,
776	14		0314 + 2, . . . laden
777	Ø3		
778	99	STA 0400, Y	Akkumulatorinhalt nach Adressen 0400 + 0, 0400 + 1,
779	ØØ		0400 + 2 abspeichern
780	Ø4		
781	E8	INX	X-Register um den Wert 1 erhöhen
782	C8	INY	Y-Register um den Wert 1 erhöhen
783	CØ	CPY 1A	Inhalt des Y-Registers mit Wert 1A hex = 26 dez
784	1A		vergleichen
785	DØ	BNE F4	Verzweigen nach 775, wenn Registerinhalt #1A
786	F4		

3. Rücksprung ins aufrufende Programm

787	6Ø	RTS	

4. Datenliste

788	C1	A	ASCII-Codes des Alphabets
789	C2	B	
790	C3	C	
791	C4	.	
792	C5	.	
793	C6	.	
794	C7	.	
795	C8	.	
796	C9	.	
797	CA	.	
798	CB	.	
799	CC	.	
800	CD	.	
801	CE	.	
802	CF	.	
803	DØ	.	
804	D1	.	
805	D2	.	
806	D3	.	
807	D4	.	
808	D5	.	
809	D6	.	
810	D7	.	
811	D8	.	
812	D9	Y	
813	DA	Z	

Einzelne Programmschritte werden hier näher erläutert:

SCHRITT 1

Neben der bereits bekannten Löschung des Bildschirms mit JSR FC58 werden in diesem Programmabschnitt die beiden Indexregister Y, X für ihre nachfolgende Aufgabe mit den Anfangswerten 00 hex geladen.

SCHRITT 2

Dieser Programmabschnitt liest aus dem Speicherbereich 0314 . . . 032D hex oder 788 . . . 813 dez 26 ASCII-Codes und speichert sie der Reihe nach im Bildschirmspeicher unter den 26 Adressen 0400 . . . 041A hex ab. Das Ergebnis ist die Darstellung der Buchstaben A . . . Z auf der ersten Bildschirmzeile.

Adressen 775, 776, 777. Mit dem Code BD hex wird eine Version des LDA-Befehls gewählt, die den Inhalt der Speicherstelle mit der absoluten Adresse 0314 + X in den Akkumulator A lädt. Zur absoluten Adresse 0314 hex wird als Adresszeiger der Inhalt des X-Registers addiert. Bei der erstmaligen Ausführung ist X = 00 hex; der Akkumulator wird also mit ASCII-Code C1 hex oder dem Buchstaben A geladen.

Adressen 778, 779, 780. Die Funktion dieses Maschinenbefehls ist wie im vorigen Programmbeispiel.

Adressen 781 und 782. Nach jeder Zeichenübertragung (Datenliste—Akkumulator—Bildschirmspeicher) wird der Adreßzeiger durch Inkrementieren der Register X und Y um den Wert 1 weitergerückt. Beim zweiten Schleifendurchlauf beispielsweise haben sich die absoluten Anfangsadressen (0314, 0400) in den Befehlen (LDA, STA) verändert in (0314 + 01 = 0315, 0400 + 01 = 0401).

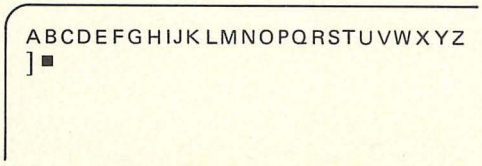
Adressen 783, 784. Auch die Funktion dieses Vergleichsbefehls ist wie zuvor. Die Zahl der Schleifendurchläufe läßt sich am Inhalt des Y-Registers ablesen; sein Wert wird mit dem vorgegebenen Wert 1A hex oder 26 dez verglichen.

Adressen 785, 786. Mit dem Code D0 wird wie zuvor ein Verzweigungsbefehl angegeben, der zur Adresse 775 zurückverzweigt, solange der vorangehende Vergleich CPY 1A nicht Gleichheit feststellt.

SCHRITT 4

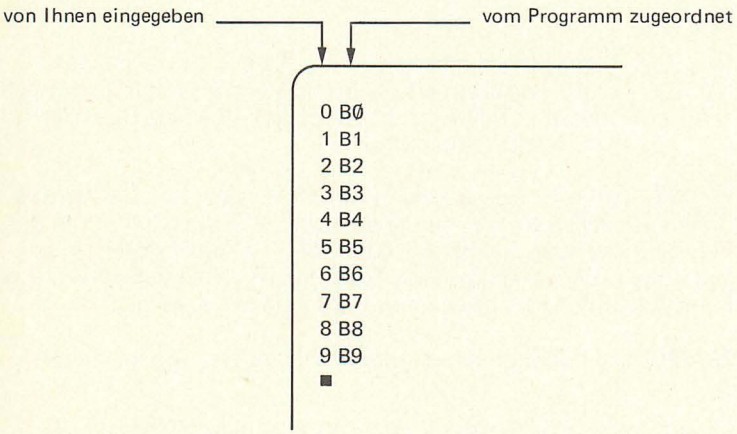
Hier die nach dem eigentlichen Maschinenprogramm liegende Datenliste mit den 26 ASCII-Codes des Alphabets (siehe auch Anhang C–2).

Nach Ablauf des Maschinenprogramms finden Sie auf Ihrem Bildschirm folgende Darstellung:



ASCII-Codes darstellen

In diesem Beispiel zeigen wir erstmals, wie über die Tastatur Daten in ein laufendes Maschinenprogramm eingegeben werden können. Das Programm stellt neben dem von Ihnen eingegebenen Zeichen den zugehörigen ASCII-Code auf dem Bildschirm dar:



Zum Einlesen von Tastenanschlägen in das Maschinenprogramm und zur Darstellung von Inhalten des Akkumulators A als Hexadezimalzahlen auf dem Bildschirm benutzen wir zwei weitere, fest im APPLE-Computer eingespeicherte Maschinenprogramme:

Anfangsadresse im Speicher		Funktion
dez	hex	
64 821	FD35	TASTENEINGABE LESEN
65 517	FDED	AKKUMULATORINHALT IM ASCII-CODE DARSTELLEN

Außerdem zeigt das Programm, wie sich mit Maschinenbefehlen der Cursor auf dem Bildschirm verschieben und der Sprung von einer Bildschirmzeile an den Anfang der nächsten realisieren lassen.

Die Programmeingabe über das Betriebssystem BBS beginnt mit folgenden ANGABEN ZUM MASCHINENPROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 41

Hier die einzugebenden Codes des **Maschinenprogramms zur Darstellung der ASCII-Codes:**

1. Initiieren

```
768 A2      LDX 0          Zähler (X-Register) auf Wert 0 setzen
769 00
770 20      JSR FC58
771 58
772 FC
```

2. Tasteneingabe lesen

```
Start
773 20      JSR FD35       Sprung in Maschinenprogramm "TASTENEINGABE
774 35                     LESEN" unter Anfangsadresse FD35
775 FD
776 8D      STA 0340       Gelesenes Zeichen unter 0340 abspeichern
777 40
778 03
779 20      JSR FDED       Sprung in Maschinenprogramm "ZEICHEN DAR-
780 ED                     STELLEN" unter Anfangsadresse FDED
781 FD
```

3. Leerzeichen darstellen

```
782 A9      LDA A0         ASCII-Code A0 für 'Leerzeichen' laden
783 A0
784 20      JSR FDED       'Leerzeichen darstellen' = Cursor bewegen
785 ED
786 FD
787 A9      LDA A0
788 A0                     Erneute Cursorbewegung um 1 Spalte nach rechts
789 20      JSR FDED
790 ED
791 FD
```

4. ASCII-Code darstellen

```
792 AD      LDA 0340       Gelesenes Zeichen vom Speicher zurückholen
793 40
794 03
795 20      JSR FDDA       Zeichen darstellen
796 DA
797 FD
```

5. Sprung auf neue Zeile

```
798 A9      LDA 8D         ASCII-Code für 'Sprung auf folgenden Zeilenanfang'
799 8D                     laden
800 20      JSR FDED       'Sprung auf folgenden Zeilenanfang' ausführen
801 ED
802 FD
```

6. Programmschleife wiederholen/abbrechen

```
803 E8      INX            X-Register inkrementieren
804 E0      CPX 1A         Inhalt von Register X mit Wert 1A hex = 26 dez
805 1A                     vergleichen
806 D0      BNE DD         Verzweigen nach 773, wenn Registerinhalt ≠ 1A,
807 DD                     und neues Zeichen lesen
```

7. Rücksprung ins aufrufende Programm

```
808 60      RTS
```

Hier die Erklärungen zu den einzelnen Programmschritten.

SCHRITT 1

Adressen 768, 769. Mit dem unmittelbaren Ladebefehl LDX 0 wird das X-Register für seine spätere Funktion als Zähler (der 26 erwarteten Eingaben) auf den Anfangswert 00 hex gesetzt.

Die folgenden Schritte, 2, 3, 4, 5, 6 bilden die Befehle einer Programmschleife, die 26mal entsprechend den erwarteten 26 Zeicheneingaben durchlaufen wird.

SCHRITT 2

Die folgenden drei Maschinenbefehle JSR, STA, JSR bilden die Übertragungskette Tastatur→Akkumulator→Bildschirm.

Adressen 773, 774, 775. Mit JSR FD35 springt das Maschinenprogramm in das fest eingespeicherte Unterprogramm ZEICHEN EINLESEN mit der absoluten Anfangsadresse FD35 hex. Dieses Unterprogramm nimmt jedes am Tastenfeld eingegebene Zeichen entgegen und speichert es in den Akkumulator A.

Adressen 776, 777, 778. Das gerade in den Akkumulator übertragene Zeichen (genauer: sein ASCII-Code) wird mit STA 0340 und unter der absoluten Speicheradresse 0340 hex oder 832 dez zwischengespeichert. (In Schritt 4 wird es von dort in den Akkumulator zurückgeholt und in Form seines ASCII-Codes auf dem Bildschirm dargestellt.)

Adressen 779, 780, 781. Mit dem Sprungbefehl JSR FDED wird das bereits bekannte Maschinenprogramm ZEICHEN DARSTELLEN im APPLE-Computer angesprochen. Es liest den ASCII-Code aus dem Akkumulator A und stellt ihn durch sein ursprüngliches Zeichen auf dem Bildschirm dar.

SCHRITT 3

Die folgenden Doppelbefehle LDA, JSR und LDA, JSR verschieben den Cursor um zwei Leerstellen nach rechts (wo anschließend die Darstellung des ASCII-Codes erfolgen soll).

Adressen 782, 783. Mit dem absoluten Ladebefehl LDA A0 wird der ASCII-Code A0 hex für "Leerstelle" in den Akkumulator geladen.

Adressen 784, 785, 786. Mit dem Sprung JSR FDED in das fest eingespeicherte Maschinenprogramm ZEICHEN DARSTELLEN wird das Leerzeichen "dargestellt", d.h. der Cursor verschoben.

SCHRITT 4

Die folgenden Befehle LDA und JSR holen den ASCII-Code des zwischengespeicherten Zeichens aus seiner Speicherstelle 832 dez und stellen ihn als ASCII-Code dar, d.h. als zweistellige Hexadezimalzahl entsprechend der Tabelle in Anhang C–3.

Adressen 795, 796, 797. Mit dem Sprung JSR FDDA wird ein fest eingespeichertes Maschinenprogramm angesprochen, das den ASCII-Code im Akkumulator A liest und als ASCII-Code auf dem Bildschirm darstellt.

SCHRITT 5

In den folgenden beiden Befehlen LDA und JSR wird der ASCII-Code 8D für "Sprung auf den Anfang der folgenden Bildschirmzeile" in den Akkumulator gelesen und vom fest eingespeicherten Maschinenprogramm ZEICHEN DARSTELLEN ausgeführt.

SCHRITT 6

Die folgenden Befehle INX, CPX, BNE benutzen das X-Register als Zähler für die bereits erfolgten Schleifendurchläufe und eingelesenen Zeichen und beenden den Einlesevorgang, sobald 26 Zeichen eingegeben wurden.

Adresse 803. Mit INX wird der Inhalt des X-Registers um den Wert 1 erhöht.

Adressen 804, 805. Mit dem Operationscode E0 hex wird eine Version des Vergleichsbefehls CPX bezeichnet, die den Inhalt des X-Registers mit dem unmittelbar nach dem Maschinenbefehl E0 folgenden Zahlenwert (hier 1A hex oder 26 dez) vergleicht. Zum Vergleich wird vom Inhalt des X-Registers der angegebene Wert 1A hex subtrahiert und beim Ergebnis Null das Bit Z im Statusregister des Mikroprozessors gesetzt.

Adressen 806, 807. Der relativ verzweigende Befehl BNE DD macht eine Verzweigung vom Ergebnis des vorangegangenen Vergleichs abhängig. Ist $X \neq 26$ dez, dann setzt dieser Befehl den Programmzähler von seinem augenblicklichen Wert 808 dez auf den Wert 773 dez um –35 dez Schritte zurück. (–35 dez wird nach der Tabelle in Anhang B hexadezimal durch DD dargestellt).

Dieses Programm hat den gleichen Wert wie die ASCII-Tabelle im Anhang dieses Buchs: Zu jedem eingegebenen Zeichen teilt es Ihnen den zugehörigen ASCII-Code mit. Sie können die Zahl erforderlicher Eingaben durch Ändern des Hexadezimalwerts in Adresse 805 verringern.

Mehrzeilige Textdarstellungen

Dieses Maschinenprogramm liefert auf dem Bildschirm folgende Textwiedergabe:

MEHR–
ZEILIGER
TEXT

Dieser Test ist in einer Datenliste nach dem eigentlichen Maschinenprogramm abgelegt und an geeigneten Stellen mit dem ASCII-Code 8D hex für "Sprung auf den Anfang der folgenden Bildschirmzeile" versehen.

Für die Eingabe des Programms hier die ANGABEN:

Anfangsadresse: 768
Anzahl der Bytes: 39

Hier die Codes des **Maschinenprogramms für mehrzeilige Textdarstellungen:**

1. Initiieren

```
768 20      JSR FC58
769 58
770 FC

771 A2      LDX 0
772 00
```

2. Zeichen lesen

```
773 BD      LDA 0311, X
774 11
775 03

776 20      JSR FDED
777 ED
778 FD

779 E8      INX

780 E0      CPX 13        Vergleich des Registerinhalts X mit dem Wert 13 hex
781 13                    = 19 dez (Umfang der Datenliste)

782 D0      BNE F5        Verzweigen nach 773, wenn Registerinhalt ≠13
783 F5
```

3. Rücksprung ins aufrufende Programm

```
784 60      RTS
```

4. Datenliste

```
785 CD      M
786 C5      E
787 C8      H
788 D2      R
789 2D      –
790 8D      ← Sprung auf folgenden Zeilenanfang
791 DA      Z
792 C5      E
793 C9      I
794 0C      L
795 C9      I
796 C7      G
797 C5      E
798 D2      R
799 8D      ← Sprung auf folgenden Zeilenanfang
800 D4      T
801 C5      E
802 D8      X
803 D4      T
```

Es folgen Erklärungen zu den Programmschritten:

SCHRITT 1

Für die Bildung eines Adreßzeigers zur Datenliste nach dem Maschinenprogramm wird das X-Register benutzt, das hier auf seinen Anfangswert 00 hex gesetzt wird.

SCHRITT 2

Die Befehlsfolge LDA . . . BNE bildet eine Programmschleife, die entsprechend der Länge der Datenliste unter Adressen 785 . . . 803 dez oder 0311 . . . 0323 hex 19mal

durchlaufen wird.

Mit LDA 0311, X werden die Zeichen der Datenliste der Reihe nach in den Akkumulator A geladen und mit dem anschließend durch JSR FDED aufgerufenen Maschinenprogramm ZEICHEN DARSTELLEN auf dem Bildschirm wiedergegeben. Zur absoluten Anfangsadresse 0311 hex wird bei jeder Befehlsausführung der Inhalt des X-Registers als Adreßzeiger addiert.

Mit INX wird der Inhalt des X-Registers um den Wert 1 erhöht und durchläuft dabei in unserem Beispiel die Werte 00 . . . 13 hex oder 0 . . . 19 dez.

Der Inhalt des X-Registers wird bei jedem Schleifendurchlauf im Vergleichsbefehl CPX 13 mit der gesetzten Grenze 13 hex verglichen.

Sind noch nicht alle Zeichen der Datenliste gelesen (wobei zu den Zeichen auch die beiden Zeichen "Sprung auf den Anfang der folgenden Bildschirmzeile") gehören, dann erfolgt mit BNE F5 der relative Rücksprung zur Adresse 773.

SCHRITT 4

Bemerkenswert am Aufbau dieser Datenliste für eine Textdarstellung ist, daß sie als gleichwertiges Textzeichen auch den ASCII-Code **8D hex** für **"Sprung auf den Anfang der folgenden Bildschirmzeile"** enthält.

Inverse Zeichendarstellungen

Bis jetzt haben wir in unseren Bildschirmdarstellungen mit leuchtenden Zeichen auf einem dunklen Hintergrund gearbeitet. Wir können auch in den Darstellungsmode "invers" übergehen, in dem der Hintergrund hell aufleuchtet und das Zeichen dunkel bleibt.

Zum Übergang auf inverse Darstellungen muß lediglich ein ASCII-Code verwendet werden, in dem die beiden höchstwertigen Binärziffern 11 durch die Binärziffer 00 ersetzt wurden:

ASCII-Code	Binärdarstellung	
C1	1 1 0 0 0 0 0 1	normal
01	0 0 0 0 0 0 0 1	invers
41	0 1 0 0 0 0 0 1	blinkend

↑↑
diese Bits entscheiden über
normal / invers / blinkend

Machen wir einen Versuch und stellen im vorangegangenen Programm das Wort TEXT invers dar. Hierzu sind folgende Änderungen erforderlich:

```
800  14
801  05
802  18
803  14
```

TEXT invers

Die ASCII-Codes für inverse Textdarstellungen finden wir in Anhang C–2. Hier ein Auszug für die Buchstaben A . . . Z:

Zeichen		Darstellungsart		
		normal	invers	blinkend
A		C1	Ø1	41
B		C2	Ø2	42
C		C3	Ø3	43
D		C4	Ø4	44
E		C5	Ø5	45
F		C6	Ø6	46
G		C7	Ø7	47
H		C8	Ø8	48
I		C9	Ø9	49
J		CA	ØA	4A
K		CB	ØB	4B
L	Code	CC	ØC	4C
M		CD	ØD	4D
N		CE	ØE	4E
O		CF	ØF	4F
P		DØ	1Ø	5Ø
Q		D1	11	51
R		D2	12	52
S		D3	13	53
T		D4	14	54
U		D5	15	55
V		D6	16	56
W		D7	17	57
X		D8	18	58
Y		D9	19	59
Z		DA	1A	5A

Blinkende Darstellung

Eine weitere Darstellungsmöglichkeit sind blinkende Zeichen.

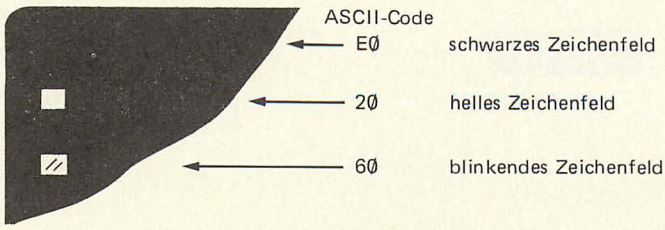
Zum Übergang von normalen Darstellungen zu blinkenden wird im ASCII-Code der Zeichen das höchstwertige Bit 1 durch das Bit 0 ersetzt.

Das Wort TEXT im vorigen Programmbeispiel stellen wir durch folgende Änderungen blinkend dar:

```
800  54
801  45
802  58
803  54
```

TEXT blinkend

Mit den ASCII-Codes 20 und 60 hex erhalten wir ein wie der Cursor hell aufleuchtendes oder blinkendes Zeichenfeld:



Texteingabe mit Bildschirmdarstellung

Das folgende Maschinenprogramm ist das einfachste und eindruckvollste Programm in diesem Kapitel "Text". Es gestattet Ihnen eine fortlaufende Texteingabe zur Darstellung auf dem Bildschirm.

Durch Eingabe eines Schrägstrichs "/" können Sie die Texteingabe beenden.

Für den Dialog mit dem Betriebssystem BBS hier die ANGABEN ZUM MASCHINENPROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 17

Hier die einzugebenden Codes des **Maschinenprogramms für Texteingaben:**

1. Bildschirm löschen

```
768 20      JSR FC58
769 58
770 FC
```

2. Zeichen lesen und darstellen

```
771 20      JSR FD35      Tasteneingabe lesen und im Akkumulator speichern
772 35
773 FD

774 C9      CMP AF        Akkumulatorinhalt mit Code AF für "/" vergleichen
775 AF

776 F0      BEQ 06        Verzweige nach 784, wenn Akkumulatorinhalt = AF
777 06

778 20      JSR FDED      Zeichen darstellen
779 ED
780 FD

781 4C      JMP 0303      Sprung zu Adresse 0303 hex = 771 dez
782 03
783 03
```

3. Rücksprung ins aufrufende Programm

```
784 60      RTS
```

Hier Erläuterungen zu einem Programmschritt:

SCHRITT 2

In diesem Programmabschnitt bilden die Befehle JSR FD35, JSR FDED, JMP 0303 eine endlose Programmschleife, die eingegebene Tastenzeichen unmittelbar zum Bildschirm weiterreicht.

Mit den Befehlen CMP AF, BEQ 06 wird der durchlaufende Zeichenstrom auf das Endezeichen "/" abgefragt. Tritt dieses Zeichen auf, dann verzweigt BEQ 06 zum Programmende.

Adressen 771, 772, 773. Mit JSR FD35 wird das festeingespeicherte Maschinenprogramm ZEICHEN LESEN aktiviert. Es überträgt Tastenzeichen von der Tastatur zum Akkumulator.

Adressen 778, 779, 780. Mit JSR FDED wird das festeingespeicherte Maschinenprogramm ZEICHEN DARSTELLEN aktiviert. Es stellt den Inhalt des Akkumulators als

Zeichen auf dem Bildschirm dar, wobei es einen Akkumulatorinhalt **8D hex** als **"Sprung auf den Anfang der nächsten Bildschirmzeile"** und den Speicherinhalt **A0 hex** als **"Leerzeichen"** versteht.

Adressen 781, 782, 783. Mit dem Operationscode 4C hex wird eine Version des Sprungsbefehls JMP bezeichnet, die einen unbedingten Sprung zu einer absoluten Adresse auslöst, deren beiden Bytes unmittelbar nach dem Befehl angegeben sind. In diesem Fall 0303 hex oder 771 dez.

Adressen 774, 775. Mit dem Operationscode C9 hex wird eine Version des Vergleichsbefehls CMP bezeichnet, die den Inhalt des Akkumulators mit dem Inhalt des unmittelbar nach dem Befehl stehenden Byte, hier AF hex, vergleicht. Tatsächlich subtrahiert dieser Befehl den Wert AF vom Inhalt des Akkumulators und setzt das Bit Z im Statusregister des Mikroprozessors, abhängig vom Ergebnis. Der Wert AF hex ist der ASCII-Code für "Schrägstrich".

Adressen 776, 777. Der Verzweigungsbefehl BEQ 06 verzweigt zum Programmende nach Adresse 784 dez, wenn der vorangehende Vergleich im Zeichenstrom das Zeichen "Schrägstrich /" gefunden hat.

Vielleicht ist Ihnen das Endezeichen "Schrägstrich" zu gefährlich, da Sie es versehentlich eingeben können und vielleicht wollen Sie daher ein anderes Endezeichen vorschreiben.

Bei Texteingaben über die 24 Zeilen des Bildschirms hinaus, werden die oberen Bildschirmzeilen "aufgerollt".

8
TON - und BBS

Einführung

Der APPLE-Computer kann Auge und Ohr ansprechen. Wir werden in den folgenden Beispielen lernen, wie wir mit Maschinenbefehlen und dem Lautsprecher im APPLE-Computer eine Tonleiter bauen können.

Jeder Maschinenbefehl, der die absolute Adresse C030 hex verwendet, "zupft" an der Lautsprechermembran. Je häufiger wir diese Adresse ansprechen, umso höher die abgestrahlte Frequenz. Je länger wir diese Adresse ansprechen, umso größer die Tondauer.

Die folgenden Maschinenprogramme zeigen, wie man Tonfrequenz und Tondauer durch Eingaben vom Tastenfeld wählen kann – und damit den APPLE-Computer in ein elektronisches Musikinstrument verwandelt.

Erweiterung des BBS

Unser Maschinenprogramm "Tonexperimente" läßt uns die Wahl zwischen 256 verschiedenen Tonfrequenzen. Um diese 256 Frequenzen hören zu können, wollen wir das Maschinenprogramm beliebig oft ablaufen lassen – jedesmal mit einer neuen Tonfrequenz. Im Betriebssystem BBS haben wir zwischen den Programmzeilen 700 . . . 900 genügend Platz für BASIC-Befehle, die den Ablauf des Tonprogramms mit neuen Werten für die Tonfrequenz wiederholen.

```
700 REM * PROGRAMM STARTEN *
 .
 .
 .
800 CALL S
 .
 .
 .
900 END
```

Hier die BASIC-Befehle, nach deren Einfügung in das Betriebssystem BBS Sie die folgenden "Tonexperimente" beliebig oft mit neuen Tonhöhen wiederholen können.

```
700  REM * PROGRAMM STARTEN *
710  HOME
720  PRINT "TONHOEHE.............1
     -255": PRINT
725  PRINT "ERNEUTE KORREKTUR...0
     ": PRINT
730  PRINT "PROGRAMMENDE........9
     9": PRINT
735  INPUT P
740  IF P = 0 THEN  GOTO 430
745  IF P = 99 THEN  GOTO 900
750  POKE 0,P
800  CALL S
810  GOTO 720
860  PRINT : PRINT : PRINT
870  PRINT "WIDERHOLUNG MIT NEUEN
      WERTEN? J/N "
880  GET C$: IF C$ <  > "J" AND C
     $ <  > "N" THEN  GOTO 870
890  IF C$ = "J" THEN  GOTO 430
900  END
```

Tonexperimente

Das folgende Maschinenprogramm bildet zusammen mit dem Lautsprecher im APPLE-Computer einen Tongenerator variabler Tonfrequenz und fester Tondauer.

Die Tonfrequenz wird aus der Speicherstelle mit der Adresse 0000 hex abgeholt – wo sie zuvor der BASIC-Befehl POKE 0, P im aufrufenden Betriebssystem BBS abgelegt hatte (Programmzeile 750).

Die Programmeingabe beginnt mit folgenden ANGABEN ZUM MASCHINENPROGRAMM.

Anfangsadresse: 768
Anzahl der Bytes: 26

Hier die einzugebenden Codes eines **Maschinenprogramm für Tonexperimente:**

1. Initiieren

768 A9 769 C8	LDA C8	Tondauer durch Wert C8 bestimmen
770 85 771 Ø1	STA 0001	Tondauer unter 0001 abspeichern
772 2Ø 773 58 774 FC	JSR FC58	Bildschirm löschen

```
2. Lautsprecher ertönen lassen
   775 AD      LDA C030     "Anzupfen des Lautsprechers"
   776 30
   777 C0
   778 88      DEY          Y-Register dekrementieren
   779 D0      BNE 04       Verzweigen nach 785, wenn Registerinhalt ≠0
   780 04
   781 C6      DEC 0001     Speicherinhalt zu Adresse 0001 dekrementieren
   782 01                   (Wert C8 für Tondauer)
   783 F0      BEQ 08       Verzweigen nach 793, wenn Speicherinhalt = 0
   784 08
   785 CA      DEX          X-Register dekrementieren
   786 D0      BNE F6       Verzweigen nach 778, wenn Registerinhalt ≠0
   787 F6
   788 A6      LDX 0000     X-Register mit Speicherinhalt von Adresse 0000 laden
   789 00
   790 4C      JMP 0307     Sprung nach 0307 hex = 775 dez
   791 07
   792 03
3. Rücksprung
   793 60      RTS
```

SCHRITT 1

Die beiden Befehle LDA, STA übertragen einen festen Zahlenwert (hier C8 hex) für die Tondauer in den Akkumulator und von dort in die Speicherstelle mit der absoluten Adresse 0001. Durch Ändern des Zahlenwerts C8 hex ändert sich die Tondauer.

SCHRITT 2

Die Befehlsfolge dieses Programmabschnitts bildet das eigentliche Herz des Tongenerators: Hier werden eingegebene Zahlenwerte in tatsächliche Tondauer und Periodendauer umgewandelt (Periodendauer ist der zeitliche Abstand zwischen wiederholtem "Anzupfen" der Lautsprechermembran. Sie bestimmt die Tonfrequenz).

Das Prinzip der Bildung von Verzögerungszeiten (wie Tondauer, Periodendauer) besteht in Programmschleifen, die einen Zähler enthalten. Bei jedem Schleifendurchlauf wird der Zähler um den Wert 1 dekrementiert. Hat er den Wert 0 erreicht, dann wird im Fall "Tonfrequenz" der Lautsprecher angezupft und der Zähler auf seinen Anfangswert zurückgesetzt, im Fall "Tondauer" das Programm beendet.

Der Zähler für die Tondauer ist das Y-Register. Der Zähler für die Tonfrequenz ist das X-Register. Beide Zähler sind zu einer Programmschleife zusammengefaßt, zu der die Befehle DEY, DEX, BNE gehören.

Die Anfangswerte beider Zähler (X, Y) könnten in Schritt 1 auf FF hex gesetzt werden; unser Ohr nimmt jedoch Unregelmäßigkeiten durch andere Anfangswerte nicht wahr.

Tondauer

Als Zähler für die Tondauer dient die Speicherstelle mit der Adresse 0001 hex.

Im Schritt 1 wurde als Anfangswert dieses Zählers C8 hex eingespeichert. Wegen der großen zeitlichen Verschiedenheit von Tondauer und Periodendauer wird der Zähler für die Tondauer nicht genauso häufig dekrementiert wie der Zähler für die Periodendauer (das X-Register). Als "Teiler" der häufigen Schleifendurchläufe in der Befehlsfolge DEY, DEX, BNE dient das Y-Register: Nach jeweils 256 Dekrementierungen des Y-Registers erfolgt 1 Dekrementierung des Zählers für die Tondauer.

Adresse 778. Mit DEY wird das Y-Register um den Wert 1 dekrementiert.

Adressen 779, 780. Der Verzweigungsbefehl BNE 04 überspringt den nachfolgenden Befehl zur Dekrementierung des Zählers für die Tondauer, wenn das Y-Register nicht den Wert 0 enthält. Dieser Wert tritt nach jeweils 256 Dekrementierungen des Y-Registers auf. Dann verzweigt der Befehl BNE nicht und ermöglicht im nachfolgenden Befehl die Dekrementierung des Zählers "Tondauer".

Adressen 781, 782. Mit dem Operationscode C6 wird eine Version des Dekrementierungsbefehls DEC bezeichnet, in der nur Speicherstellen im Adreßbereich 0000 . . . 00FF hex jeweils um den Wert 1 dekrementiert werden. Dieser Befehl gestattet zusammen mit seinen anderen Versionen, jeder beliebigen Speicherstelle in der Nullseite die Funktion eines Zählers zu geben, der schrittweise dekrementiert wird. In diesem Fall wird der Anfangswert C8 hex auf den Wert 00 hex dekrementiert – und die Dauer dieses Dekrementierungsvorgangs bestimmt die Tondauer.

Adressen 783, 784. Hat die Dekrementierung des Zählers "Tondauer" den Wert 00 hex erreicht, dann verzweigt der Befehl BEQ 08 um +8 Programmschritte nach vorne, das heißt zum Programmende in Adresse 793. Damit wird die Abstrahlung dieses Tons beendet.

Tonfrequenz

Adresse 785. Der Befehl DEX dekrementiert bei jedem Schleifendurchlauf das X-Register um den Wert 1. Das X-Register dient als Zähler für die Tonfrequenz.

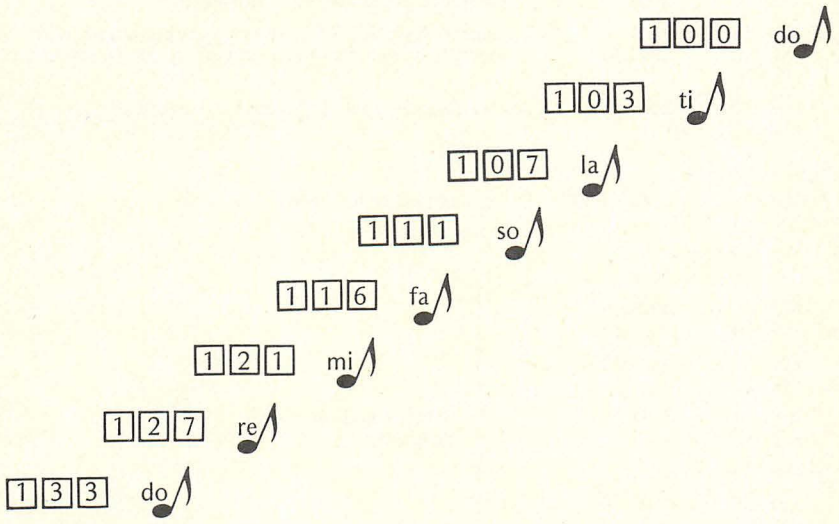
Adressen 786, 787. Hat das X-Register nicht den Wert 00 hex erreicht, dann verzweigt der Befehl BNE F6 an den Anfang der Programmschleife zu Adresse 778 zurück. Andernfalls werden die beiden folgenden Befehle LDX, JMP ausgeführt.

Adressen 788, 789. Der Zähler "Tonfrequenz" wird mit dem Befehl LDX 0000 wieder mit seinem Anfangswert geladen. Dieser Anfangswert war zuvor im aufrufenden Betriebssystem BBS mit dem BASIC-Befehl POKE 0, P unter der Adresse 0000 hex abgelegt worden. Der Anfangswert des Zählers (X-Register) kann im Bereich 0 . . . 255 dez liegen. Beim Wert 255 dauert das Dekrementieren des X-Registers am längsten – und folglich strahlt hier der Lautsprecher die tiefste Frequenz ab.

Adressen 790, 791, 792. Mit dem Sprungbefehl JMP 0307 zur absoluten Adresse 0307 hex oder 775 dez erfolgt der Sprung zu einem Maschinenbefehl, der die Adresse C030 hex anspricht (LDA C030) und damit den Lautsprecher "anzupft". Dieser Vorgang wiederholt sich erst, wenn in der Programmschleife mit DEY, DEX, BNE das X-Register wieder den Wert 0 angenommen hat.

Die programmierte Tonleiter

Nachdem Sie die Tonfrequenzen einer Oktave durch Experimente gefunden haben und ihnen Zahlen zuordnen können, ist es leicht, diese Zahlen in einem Maschinenprogramm zu speichern und sie sich vorspielen zu lassen. Hier die Zahlen, die wir gefunden haben.



Im folgenden Maschinenprogramm arbeiten wir erstmals mit dem Stapelspeicher und den Befehlen PHA (PUSH ACCUMULATOR ON STACK) und PLA (PULL OFF STACK TO ACCUMULATOR).

Die Programmeingabe über das BBS beginnt mit folgenden ANGABEN ZUM MASCHINENPROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 54

Hier das **Maschinenprogramm für die programmierte Tonleiter:**

1. Initiieren

768 20 769 58 770 FC	JSR FC58	Bildschirm löschen
771 A9 772 C8	LDA C8	Tondauer laden
773 85 774 01	STA 0001	Tondauer abspeichern
775 A2 776 00	LDX 00	Zähler auf Wert 0 setzen

Fortsetzung ⟶

2. Tonhöhe laden

→	777 BD	LDA 032E,X	Tonhöhe in Akkumulator laden; Adressen 032E+0,
	778 2E		032E+1, . . . hex oder 814+0, 814+1, . . . dez der
	779 Ø3		Datenliste
	780 85	STA 0000	Tonhöhe unter 0000 abspeichern
	781 ØØ		
	782 E8	INX	Adreßzeiger erhöhen (für nächste Tonhöhe in der Datenliste)
*	783 8A	TXA	Adreßzeiger in Akkumulator laden
*	784 48	PHA	Adreßzeiger auf Stapelspeicher zwischenspeichern.
	785 EØ	CPX 08	Vergleich mit maximalem Umfang der Datenliste (8)
	786 Ø8		
	787 FØ	BEQ 17	Verzweigen nach 812, wenn X-Registerinhalt = 08
	788 17		

3. Ton wiedergeben

→	789 AD	LDA C030	'Lautsprecher anzupfen'
	790 3Ø		
	791 CØ		
	792 88	DEY	
	793 DØ	BNE 04	
	794 Ø4		
	795 C6	DEC 0001	
	796 Ø1		
	797 FØ	BEQ 08	Schleife wie im vorigen Programm
	798 Ø8		
	799 CA	DEX	
	800 DØ	BNE F6	
	801 F6		
	802 A6	LDX 0000	
	803 ØØ		
	804 4C	JMP 0315	
	805 15		
	806 Ø3		
*	807 68	PLA	Adreßzeiger vom Stapelspeicher zurückholen
*	808 AA	TAX	Adreßzeiger in das X-Register laden
	809 4C	JMP 0309	Sprung nach 0309 hex = 777 dez für neue Tonhöhe
	810 Ø9		
	811 Ø3		

4. Rücksprung ins aufrufende Programm

812 68	PLA	Letzte Eintragung im Stapelspeicher auslesen
813 6Ø	RTS	

5. Datenliste 'Tonhöhen'

814 85	In Versuchen mit dem Programm TONEXPERIMENTE gefundene Werte
815 7F	
816 79	
817 74	
818 6F	
819 6B	
820 67	
821 64	

Die einzelnen Programmschritte werden im folgenden erläutert:

SCHRITT 1

Mit JSR FC58 wird das Maschinenprogramm BILDSCHIRM LÖSCHEN angesprochen.

Die Tondauer C8 hex wird mit LDA C8 in den Akkumulator und von dort mit STA 0001 in den Speicherplatz mit der Adresse 0001 hex gebracht. (Dieser Speicherplatz dient anschließend als Zähler, dessen Anfangswert C8 mit dem Befehl DEC 0001 auf den Wert 0 dekrementiert wird).

Zum Lesen der 8 Tonfrequenzen im Speicherbereich 814 . . . 821 dient das X-Register als Adreßzeiger. Hier wird mit LDX 00 sein Anfangswert 00 gesetzt.

SCHRITT 2

Dieser Schritt ist Teil einer Programmschleife, mit der die 8 Noten in der Datenliste am Programmende gelesen und gespielt werden.

Das Lesen der 8 Noten übernimmt der LDA-Befehl mit indizierter Absolutadresse 032E+X. Diese Adresse durchläuft die Werte 032E+00 . . . 032E+08 hex oder 814 . . . 821 dez. Die gelesene Tonhöhe wird dann mit STA 0000 unter Adresse 0000 hex abgelegt.

Das X-Register dient in diesem Programmabschnitt als Indexregister, im folgenden Programmabschnitt als Zähler. Der Adreßzeiger im X-Register wird daher vorübergehend in den Stapelspeicher gerettet und vor erneuter Ausführung von Schritt 2 vom Stapelspeicher wieder in das X-Register zurückgeladen. Die Befehle TXA, PHA bilden die Übertragungskette Register X → Akkumulator A → Stapelspeicher.

Mit dem Vergleichsbefehl CPX 08 wird der Inhalt des X-Registers unmittelbar mit dem Wert 08 hex verglichen, um den folgenden Verzweigungsbefehl vorzubereiten.

Mit BEQ 17 verzweigt das Programm zum Programmende in Adresse 812, wenn der vorangegangene Vergleich Gleichheit von 08 hex (8 Notenwerte) und Inhalt des X-Registers ergeben hat.

SCHRITT 3

Dieser Programmabschnitt unterscheidet sich nur durch die drei zusätzlichen Befehle PLA, TAX, JMP von Schritt 2 des vorigen Maschinenprogramms.

Die ersten 9 Befehle bilden Programmschleifen für Dauer und Frequenz jeder der 8 Noten. Die in Schritt 2 von der Datenliste gelesene Tonhöhe wird in Adressen 802, 803 als Anfangswert zum wiederholten Laden des Zählers (X-Register) verwendet.

Adressen 807, 808. Ist die Note gespielt, dann muß die Adresse für die nächste Note in der Datenliste am Programmende gebildet werden. Mit PLA wird der im Stapelspeicher gerettete Adreßzeiger zunächst in den Akkumulator A und von dort mit TAX in das X-Register zurückgebracht. Der nachfolgende Sprung JMP 0309 zur Absolutadresse 0309 hex oder 777 dez läßt den nächsten, ausgeführten Befehl LDA 032E,X sein.

SCHRITT 4

Adresse 812. Den Befehlen CPX 08, BEQ 17 in Adressen 785 . . . 788 zum Beenden

des Tonprogramms war noch der Stapelbefehl PHA in Adresse 784 vorausgegangen. Ehe das Maschinenprogramm anschließend mit RTS verlassen wird, muß mit PLA der zuvor eingespeicherte Wert wieder ausgelesen werden.

Bei Einsprung in das Maschinenprogramm mit CALL S wurde die Speicheradresse des nächsten, auf CALL S folgenden BASIC-Befehls im Stapelspeicher abgelegt. Der Befehl RTS am Ende des Maschinenprogramms lädt den Programmzähler zur Fortsetzung des BASIC-Programms mit dieser Speicheradresse im Stapelspeicher. Ist versehentlich vom gerade abgelaufenen Maschinenprogramm eine Eintragung im Stapelspeicher liegen geblieben, dann ist die Fortsetzung des ausrufenden BASIC-Programms in Frage gestellt.

SCHRITT 5

Diese Datenliste enthält 8 Tonhöhen in Form von Hexadezimalzahlen, die in Experimenten mit dem vorigen Maschinenprogramm "Tonexperimente" gefunden wurden.

Noten am Tastenfeld spielen

Wir haben die Eingabe einer Tonhöhe durch Eingabe von Zahlen kennengelernt und ein Programm gesehen, das 8 Noten spielt. Das folgende Programm ist eine Kombination von beidem.

Wir werden 8 Tonhöhen 1 . . . 8 am Ende des Programms in einer Datenlsite speichern und jede Note mit den Tasten 1 . . . 8 spielen. Mit der Taste 0 wird das Programm beendet.

Zur Programmeingabe hier die ANGABEN ZUM MASCHINENPROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 59

Und hier das **Maschinenprogramm für Noteneingaben am Tastenfeld:**

1. Bildschirm löschen

```
768 20      JSR FC58
769 58
770 FC
```

2. Tasteneingabe lesen

```
771 20      JSR FD35      Maschinenprogramm "TASTENEINGABE LESEN",
772 35                    gelesenes Zeichen in Akkumulator bringen
773 FD

774 C9      CMP B0        Gelesenes Zeichen kleiner als B0 = '1' ?
775 B0                    (Operation: Akkumulatorinhalt minus B0)

776 30      BMI F9        Verzweigen nach 771, wenn Statusbit N = 1
777 F9                    (d.h. CMP B0 hatte MINUSERGEBNIS)

778 F0      BEQ 25        Verzweige nach 817 (Programmende)
779 25                    (d.h. CMP B0 hatte Ergebnis GLEICH NULL)

780 C9      CMP B9        Gelesenes Zeichen größer als B9 = '9' ?
781 B9

782 10      BPL F3        Verzweige nach 771, wenn Zeichen > 9
783 F3
```

3. ASCII-Code in Zahl wandeln

784 29 785 ØF	AND OF	B1 . . . B9 in Zahl 01 . . . 09 transformieren
786 AA	TAX	Zahl vom Akkumulator A → Register X laden

4. Tondauer/höhe laden

787 A9 788 6Ø	LDA 60	Tondauer laden
789 85 790 Ø1	STA 0001	Tondauer abspeichern
791 BD 792 32 793 Ø3	LDA 0332,X	Tonhöhe indiziert aus Datenliste lesen Adressen: 0332+X hex = 814+X dez
794 85 795 ØØ	STA 0000	Tonhöhe abspeichern

5. Ton wiedergeben

796 AD 797 3Ø 798 CØ	LDA C030	'Lautsprecher anzupfen'
799 88	DEY	Schleife wie in vorigen Programmen
800 DØ 801 Ø4	BNE 04	
802 C6 803 Ø1	DEC 0001	
804 FØ 805 Ø8	BEQ 08	
806 CA	DEX	
807 DØ 808 F6	BNE F6	
809 A6 810 ØØ	LDX 0000	
811 4C 812 1C 813 Ø3	JMP 031C	

6. Neue Note erwarten

814 4C 815 Ø3 816 Ø3	JMP 0303	Neue Tasteneingabe erwarten
817 6Ø	RTS	Rücksprung

7. Datenliste

818 EA	NOP	← 'Platzhalter': wirkungsloser Maschinenbefehl
819 85		← Niedrigste Tonfrequenz
820 7F 821 79 822 74 823 6F 824 6B 825 67		
826 64		← Höchste Tonfrequenz

Einzelne Programmschritte finden Sie hier erläutert:

SCHRITT 2

Dieser Programmabschnitt hat folgende Aufgaben:

- Lesen der Tasteneingaben
- Zurückweisen unzulässiger Tasteneingaben
- Beenden des Programms bei Taste 0

Adressen 771, 772, 773. Mit JSR FD35 wird das Maschinenprogramm TASTENEINGABEN LESEN angesprochen und das ASCII-Code einer eingegebenen Taste im Akkumulator A abgelegt. Für die uns interessierenden Zahlen 0 . . . 9 gilt:

Ziffern dez	ASCII-CODES hex
0	B0
1	B1
2	B2
3	B3
4	B4
5	B5
6	B6
7	B7
8	B8
9	B9

Adressen 774 . . . 777 und 780 . . . 783. Mit den Befehlen CMP B0 und CMP B9 wird der Inhalt des Akkumulators mit den zulässigen Grenzen B0, B9 der uns interessierenden ASCII-Codes verglichen. Tatsächlich wird vom Akkumulatorinhalt B0 bzw. B9 subtrahiert. Ist das Ergebnis negativ (da der eingegebene ASCII-Code im Akkumulator kleiner als B0 ist) dann verzweigt der Befehl BMI (BRANCH ON RERESULT MINUS) zurück zu Adresse 771. Ist das Ergebnis von CMP B9 positiv, (da der eingegebene ASCII-Code im Akkumulator größer ist als B9) dann verzweigt der Befehl BPL (BRANCH ON RESULT PLUS) ebenfalls zu Adresse 771.

Adressen 778, 779. Hat der Vergleich CMP B0 das Ergebnis Null ergeben, dann verzweigt der Befehl BEQ 25 durch Erhöhen des Programmzählers von Adresse 780 auf Adresse 817 zum Programmende des Maschinenprogramms.

SCHRITT 5

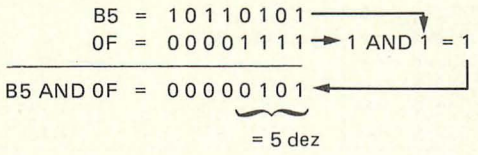
Dieser Programmabschnitt bildet zu jeder Tasteneingabe 1 . . . 8 einen Adreßzeiger, mit dessen Hilfe die zugehörige Tonhöhe aus der Datenliste am Programmende gelesen werden kann.

Ein Blick auf die ASCII-Codes B1 . . . B8 zeigt, daß die Adreßzeiger 1 . . . 8 durch Unterdrücken des "B" gewonnen werden können. Diese Operation gelingt uns mit einem logischen AND-Befehl.

Adressen 784, 785. Mit dem Code 29 hex wird eine Version des AND-Befehls gewählt, in der eine AND-Verknüpfung zwischen dem Inhalt des Akkumulators und einem unmittelbar auf den AND-Befehl folgenden Byte, hier 0F hex, erfolgt. Hier

Beispiele, die die Wirkung der AND-Verknüpfung zeigen:

Beispiel: Tasteneingabe '5' = ASCII 'B5'



Adresse 786. Als Ergebnis der vorangegangenen AND-Operation steht eine der Zahlen 01 . . . 08 hex im Akkumulator. Mit dem Befehl TAX wird dieser Wert vom Akkumulator in das Indexregister X übertragen. Im nachfolgenden Programmabschnitt macht der Befehl LDA 0332, X von diesem Inhalt des X-Registers zur Indizierung seiner absoluten Adressen Gebrauch.

```
4. TONDAUER/HÖHE LADEN
   .
   .
   .
791 BD        LDA 0332, X
792 32
793 03
   .
   .
   .
```

Adreßzeiger X	Adresse 0332, X in LDA	7. DATENLISTE
X = 0		818 EA NOP
X = 1	0332+1 = 819 dez →	819 85
X = 2	0332+2 = 820 dez →	820 7F
X = 3	0332+3 = 821 dez →	821 79
X = 4	.	822 74
X = 5	.	823 6F
X = 6	.	824 6B
X = 7	.	825 67
X = 8	0332+8 = 826 dez →	826 64

SCHRITT 4

Die Doppelbefehle LDA, STA und LDA, STA legen die Zahlenwerte für Tondauer und Tonfrequenz unter den Speicheradressen 0001 hex und 0000 hex ab.

Adressen 791, 792, 793. Der Ladebefehl LDA 0332, X hat die Aufgabe, den Wert einer Tonhöhe aus der Datenliste am Programmende in den Akkumulator zu lesen und hierbei die Tasteneingabe in Form des Registerinhalts X als Adreßzeiger zu benutzen. Hier eine Darstellung dieser Zusammenhänge:

Tasten-eingabe	ASCII-Code	nach AND	LDA 0332, X	Speicheradresse hex	Speicheradresse dez	Speicherinhalt (Note)
1	B1	01	0332+1 →	0333	819	85
2	B2	02	0332+2 →	0334	820	7F
3	B3	03	0332+3 →	0335	821	79
4	B4	04	0332+4 →	0336	822	74
5	B5	05	0332+5 →	0337	823	6F
6	B6	06	0332+6 →	0338	824	6B
7	B7	07	0332+7 →	0339	825	67
8	B8	08	0332+8 →	033A	826	64
↑ eingegeb. Zahl		↑ Zahl verwendet als Adreßindex				↑ gespielte Note

SCHRITT 5

Die Befehle dieses Programmabschnitts bilden den eigentlichen "Tongenerator" und wurden bereits vorher ausführlich beschrieben.

SCHRITT 6

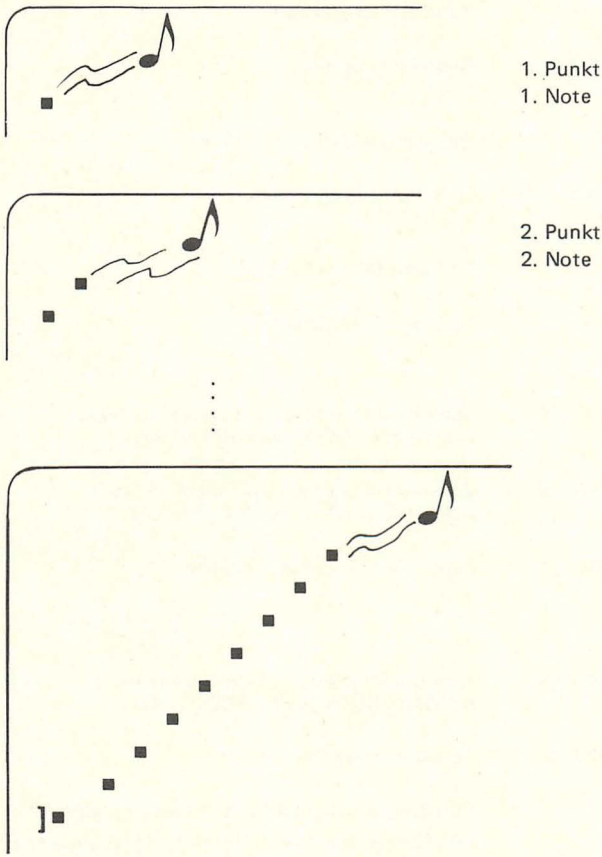
Adressen 814, 815, 816. Der unbedingte Sprungbefehl JMP 0303 bewirkt den Rücksprung an den Anfang des Programms in Adresse 771 dez. Der nachfolgende RTS-Befehl wird nur vom Verzweigungsbefehl BEQ 25 in Adressen 778, 779 erreicht.

SCHRITT 7

Die Datenliste mit den 8 Tonfrequenzen beginnt in Adresse 818 dez oder 0332 hex mit dem Befehl NOP (NO OPERATION). NOP-Befehle sind Befehle ohne Wirkung, die häufig und aus verschiedenen Gründen als bloße "Platzhalter" in Maschinenprogrammen verwendet werden. In diesem Fall belegt der NOP-Befehl einen Speicherplatz, der im Befehl LDA 0332 X im 4. Programmabschnitt adressiert – durch die Indizierung mit dem X-Register aber nicht angesprochen wird. Sollten Sie auch den Wert X = 0 zulassen, dann können Sie den Code EA hex für den NOP-Befehl durch den Hexadezimalwert einer Tonhöhe ersetzen.

Light & Sound

Zum Abschluß ein Maschinenprogramm, das nacheinander 8 Noten spielt und jede Note mit einer Bildschirmdarstellung begleitet:



Dieses Programm ist also eine Verbindung unseres Tonleiter-Programms und des Punkt-Programms aus Kapitel 6.

Zur Programmeingabe über das Betriebssystem BBS benötigen Sie folgende ANGABEN ZUM MASCHINENPROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 82

Hier die einzugebenden Codes des **Maschinenprogramms Light & Sound:**

1. Initiieren

768 A9 769 C8	LDA C8	Tondauer laden
770 85 771 Ø1	STA 0001	Tondauer abspeichern
772 2Ø 773 58 774 FC	JSR FC58	Bildschirm löschen
775 A2 776 ØØ	LDX 00	Adreßzeiger/Zähler rücksetzen
777 2Ø 778 4Ø 779 FB	JSR FB40	Graphikmode herstellen
780 A9 781 ØF	LDA 0F	Farbwert 0F = WEISS laden
782 85 783 3Ø	STA 0030	Farbwert abspeichern

2. Punkt darstellen

784 BC 785 42 786 Ø3	LDY 0342, X	Spaltenwerte indiziert aus Datenliste lesen Adressen: 0342+X hex = 834+X dez
787 BD 788 4A 789 Ø3	LDA 034A, X	Zeilenwerte indiziert aus Datenliste lesen Adressen: 034A+X hex = 842+X dez
790 2Ø 791 ØØ 792 F8	JSR F800	Punkt (Spalte, Zeile) darstellen

3. Tonhöhe laden

793 BD 794 3A 795 Ø3	LDA 033A, X	Tonhöhe indiziert aus Datenliste lesen Adressen: 033A+X hex = 826+X dez
796 85 797 ØØ	STA 0000	Tonhöhe abspeichern
798 8A	TXA	Adreßzeiger von Register X → Akkumulator A
799 48	PHA	Adreßzeiger von Akkumulator A → Stapelspeicher

4. Ton wiedergeben

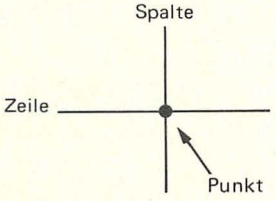
```
800 AD      LDA C030        'Lautsprecher anzupfen'
801 3Ø
802 CØ
803 88      DEY             ┐
804 DØ      BNE 04          │
805 Ø4                      │
806 C6      DEC 0001        │
807 Ø1                      │
808 FØ      BEQ 08          │
809 Ø8                      │  Schleife wie in vorigen
810 CA      DEX             │  Programmen
811 DØ      BNE F6 (–10)    │
812 F6                      │
813 A6      LDX 0000        │
814 ØØ                      │
815 4C      JUMP 0320       │
816 2Ø                      │
817 Ø3                      ┘
```

5. Alle Noten gespielt?

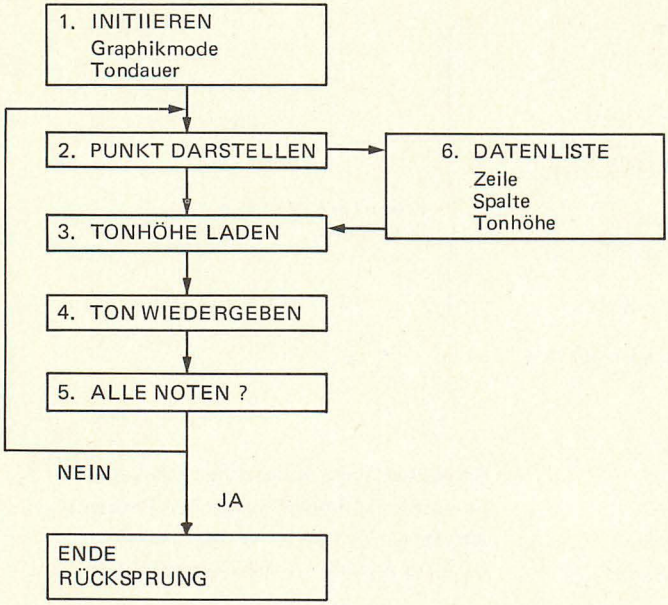
```
818 68      PLA     Adreßzeiger von Stapelspeicher → Akkumulator A
819 AA      TAX     Adreßzeiger von Akkumulator A → Register X
820 E8      INX     Adreßzeiger (X-Register) inkrementieren
821 EØ      CPX 08  Vergleich Adreßzeiger/Wert 8 (8 Noten)
822 Ø8
823 DØ      BNE D7  Verzweige nach 784, wenn Adreßzeiger ≠8
824 D7
825 6Ø      RTS     Rücksprung ins aufrufende Programm
```

6. Datenliste

```
826 88      ← Notenwerte
827 79
828 6Ø
829 67
830 59
831 51
832 48
833 44
834 Ø7      ← Spaltenwerte
835 Ø9
836 ØB
837 ØD
838 ØF
839 11
840 13
841 15
842 16      ← Zeilenwerte
843 14
844 12
845 1Ø
846 ØE
847 ØC
848 ØA
849 Ø8
```



Zum Verständnis dieses Programms (das sich aus bekannten Elementen zusammensetzt) finden Sie hier noch das Flußdiagramm:



9
ARITHMETIK - und BBS

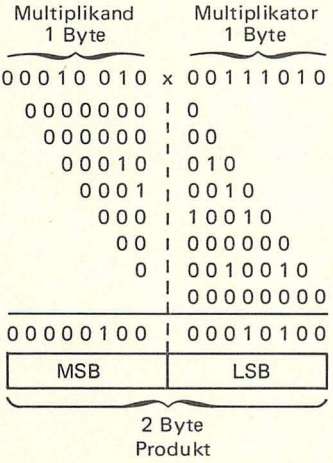
Einführung

Das Repertoire des Mikroprozessors 6502 enthält auch Befehle zur Addition und Subtraktion. Multiplikation und Division müssen durch die vorhandenen Befehle Addition und Verschiebung dargestellt werden.

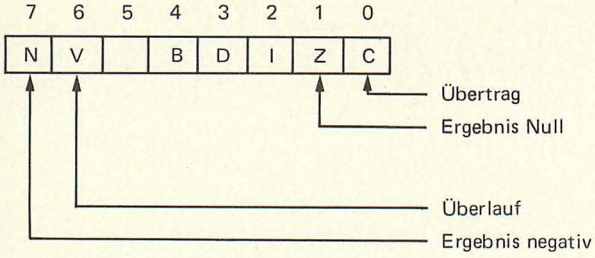
Um Operationen zu diesen vier Grundrechnungsarten im Mikroprozessor besser verfolgen zu können und Ergebnisse richtig zu interpretieren, werden wir häufig auf binäre Darstellungen zurückgreifen.

Alle Speicherplätze im APPLE-Computer fassen 8 Bits oder 1 Byte – wir arbeiten mit dem 8-Bit-Mikroprozessor 6502. Mit einem 8-Bit-Wort können wir entweder die Zahlen 0 . . . 255 dez oder –128 . . . +127 dez darstellen. Genügt dieser Zahlenumfang nicht, dann können wir Zahlen durch 2 Bytes darstellen und die arithmetischen Operationen zuerst auf das niedrigstwertige Byte, LSB, und dann auf das höchstwertige Byte, MSB, anwenden, um zum gewünschten Ergebnis zu kommen.

In 2-Byte-Arithmetik können wir einen Zahlenumfang 0 . . . 65535 dez darstellen. Zu beachten ist die Multiplikation: Hier entstehen 2-Byte-Ergebnisse, auch wenn die Ausgangszahlen (Multiplikand, Multiplikator) nur 1-Byte-Worte waren.



Eine wichtige Informationsquelle für den Ausgang arithmetischer Operationen im Mikroprozessor ist dessen Statusregister:



Von besonderer Bedeutung für uns ist das Übertragbit C: Erfordert das Ergebnis einer Addition oder Subtraktion zu seiner Darstellung mehr als die im Akkumulator A vorhandenen 8 Bits, dann wird das Übertragbit C vom Mikroprozessor auf den Wert 1 gesetzt.

Der Inhalt des Übertragbit C kann verschiedenen Zwecken dienen:

- In 1-Byte-Additionen/Subtraktionen
  Zur Aufnahme des 9. Bits der Summe/Differenz.
  Die niederwertigen 8 Bits stehen im Akkumulator.
- In 2-Byte-Additionen/Subtraktionen
  Auch zur Weitergabe eines Übertrags vom niedrigstwertigen Byte LSB zum höchstwertigen MSB.
- In Verzweigungsbefehlen wie BCC, BCS
  Zur Veränderung eines Programmablaufs in Abhängigkeit vom Ergebnis einer vorangegangenen Addition/Subtraktion.

1-Byte-Addition

Hier zunächst ein einfaches Additionsprogramm für Summen, die zu ihrer Darstellung höchstens 8 Bits erfordern und keinen Überlauf in Bit C des Statusregisters auslösen.

Wir verwenden folgenden Beispiel:

dez	hex	bin	
60	3C	00111100	← im Akkumulator
+35	+23	+ 00100011	← zum Akkumulator addiert
		1	← 1 + 0 = 1
		1	← 1 + 0 = 1
		1	← 0 + 1 = 1
		1	← 0 + 1 = 1
		1	← 0 + 1 = 1
		0	← 1 + 1 = 0 Übertrag C = 1
		1	← 0 + 0 = 0, 0 + C = 1
		0	← 0 + 0 = 0
95	5F	01011111	← Summe

Der Summand 3C hex wird mit dem Ladebefehl LDA 3C unmittelbar in den Akkumulator geladen. Der Sumand 23 hex wird mit dem Additionsbefehl ADC 23 unmittelbar zum Inhalt des Akkumulators addiert. Zur Überprüfung der Addition stellen wir die Summe hexadezimal mit dem Maschinenprogramm AKKUMULATOR-INHALT DARSTELLEN dar.

Hier die Codes eines Maschinenprogramms zur 1-Byte-Addition:

```
768 18      CLC          Statusbit C löschen
769 A9      LDA 3C       3C in Akkumulator laden
770 3C
771 69      ADC 23       23 zum Akkumulator addieren
772 23
773 2Ø      JSR FDDA     Summe (im Akkumulator) hexadezimal darstellen
774 DA
775 FD
776 6Ø      RTS          Rückkehr zum BBS
```

Die Summe steht im Akkumulator A (und hat den dortigen Summanden überschrieben). Wichtig ist der Befehl CLC zum Löschen des Übertragbit C: Der Additionsbefehl ADC berücksichtigt bei der Addition einen vorhandenen Übertrag. Der liegt hier aber nicht vor.

Überschreitet die Summe den Maximalwert 11111111, dann entsteht ein Übertrag 1, der in Bit C des Statusregisters im Mikroprozessor gespeichert wird:

```
          bin          hex
           11100000 = E0    Addend
         + 10000000 = 81    Augend
         ------------------
         1 01100001 = 161   Summe erfordert 9 Bit zur Darstellung
           |        ~~~~~~~
im         |        im
Übertrag C |        Akkumulator A
1 Bit              8 Bit
```

Das folgende erweiterte Maschinenprogramm erfaßt durch Abfrage des Übertragbit C im Statusregister auch Summen, die größer als 11111111 bin oder FF hex sind. Abhängig vom Ergebnis der Abfrage wird in diesem Programmbeispiel Bit C in die hexadezimale Summendarstellung miteinbezogen oder nicht.

Zur Programmeingabe über das Betriebssystem BBS gehören folgende ANGABEN ZUM MASCHINENPROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 22

Hier die Codes des **Maschinenprogramms einer 1-Byte-Addition:**

1. Übertragbit C löschen (C = 0)

```
768 18      CLC
```

2. Addition ausführen

```
769 A9      LDA E0          1. Summanden laden
770 E0
771 69      ADC 81          2. Summanden addieren
772 81
773 90      BCC 0B          bei C = 0 um + 11 Schritte verzweigen
774 0B
```

3. Akkumulator freimachen; Teilsumme darstellen

```
775 8D      STA 0330        niederwertige Stellen zwischenspeichern
776 30
777 03
778 A9      LDA 01          höchstwertige Stelle (Übertragbit C) laden
779 01
780 20      JSR FDDA        höchstwertige Stelle darstellen
781 DA
782 FD
783 AD      LDA 0330        niederwertige Stellen zurückholen
784 30
785 03
```

4. Teilsumme darstellen; Rückkehr

```
786 20      JSR FDDA        niederwertige Stellen darstellen
787 DA
788 FD
789 60      RTS             Rückkehr ins Hauptprogramm
```

Und hier Erklärungen zu den einzelnen Programmschritten:

SCHRITT 1

Mit dem Löschbefehl CLC (CLEAR BIT C) wird Bit C im Statusregister auf den Wert gesetzt, um nicht fälschlich einen nicht vorliegenden Übertrag in die folgende Addition einfließen zu lassen.

SCHRITT 2

Mit LDA E0 wird der erste Summand E0 hex oder 210 dez unmittelbar in den Akkumulator geladen.

Mit ADC 81 wird der Summand 81 hex oder 129 dez unmittelbar zum Akkumulatorinhalt addiert. Die entstehende Summe 339 dez oder 161 hex erfordert 9 Bits zu ihrer Darstellung. Das 9. Bit (Übertragbit) wird in Bit C des Statusregisters dargestellt.

Mit dem Verzweigungsbefehl BCC 0B erfolgt eine Programmverzweigung nur dann, wenn das Bit C im Statusregister den Wert 0 enthält. In diesem Fall werden die Befehle des folgenden Schritt 3 übersprungen und nur der Inhalt des Akkumulators als Summe dargestellt.

SCHRITT 3

Dieser Programmabschnitt wird ausgeführt, wenn mehr als 9 Bits zur Summendarstellung erforderlich sind, d.h. ein Überlauf erfolgte und Bit C auf den Wert 1 gesetzt ist.

Um auf dem Bildschirm zuerst den Inhalt von Bit C (9. Bit), dann den Inhalt des Akkumulators A (Bits 8 . . . 1) darzustellen, wird der Inhalt des Akkumulators zunächst unter Adresse 0330 hex zwischengespeichert und der Inhalt des Übertragbit C in der Form 01 hex in den Akkumulator geladen, um ihn von dort darstellen zu können.

Mit STA 0330 werden die Bits 8 . . . 1 der Summe vom Akkumulator in den Speicher unter Adresse 0330 hex zur Zwischenspeicherung übertragen.

Mit LDA 01 wird der Inhalt des Übertragbit C, der Wert 1, in Form des 8-Bit-Worts 00000001 bin oder 01 hex in den Akkumulator geladen.

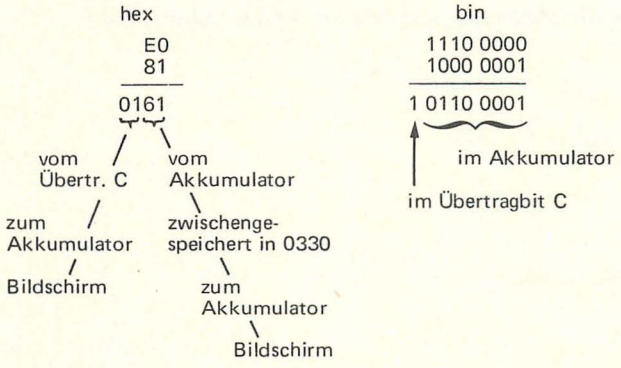
Mit dem Sprung JSR FDDA in das Maschinenprogramm AKKUMULATOR-INHALT DARSTELLEN wird der Akkumulatorinhalt hexadezimal auf dem Bildschirm dargestellt. Die Bildschirmdarstellung 01 stellt in diesem Fall das höchstwertige Byte, MSB, der Summe dar.

Mit LDA 0330 wird das niedrigstwertige Byte, LSB, der Summe von Adresse 0330 hex in den Akkumulator zurückgeladen, um im nachfolgenden Befehl (Schritt 4) auf dem Bildschirm dargestellt zu werden.

SCHRITT 4

Mit JSR FDDA erfolgt der Sprung in das Maschinenprogramm AKKUMULATOR DARSTELLEN, das den Akkumulatorinhalt hexadezimal auf dem Bildschirm darstellt.

Nach Ablauf des Programms finden Sie als Summe von E0 + 81 die Darstellung 0161 auf dem Bildschirm:

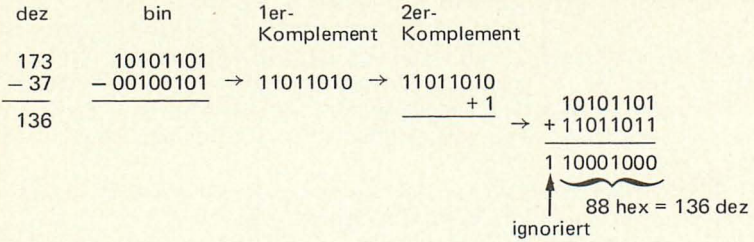


1-Byte-Subtraktion

Im folgenden Maschinenprogramm wird der Subtraktionsbefehl SBC angewandt um positive Differenzen zwischen 2 Zahlen zu bilden. Das heißt, die subtrahierte Zahl muß immer kleiner sein als der Subtrahend.

Subtraktionen werden im Mikroprozessor 6502 durch Bildung des 2er-Komple-

ments zur subtrahierten Zahl in Additionen überführt. Hier ein Beispiel:



Zur Bildung des 2er-Komplements bildet der Mikroprozessor zunächst das 1er-Komplement und addiert hierzu den Wert 1, der aus dem Übertragbit C gelesen wird. Das Subtraktionsprogramm muß daher mit dem Befehl SEC beginnen, der das Übertragbit C im Statusregister des Mikroprozessors auf den Wert 1 setzt.

Im folgenden Programmbeispiel wird zunächst der Subtrahend 3C hex oder 60 dez mit dem Befehl LDA 3C unmittelbar in den Akkumulator geladen. Von diesem Akkumulatorinhalt wird die Zahl 23 hex oder 35 dez mit dem Befehl SBC 23 unmittelbar subtrahiert. Die entstandene Differenz 19 hex oder 25 dez steht im Akkumulator und wird mit dem Sprung JSR FDDA vom Maschinenprogramm AKKUMULATORINHALT DARSTELLEN hexadezimal auf dem Bildschirm dargestellt.

Hier die ANGABEN ZUM MASCHINENPROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 9

Und hier die Codes des **Maschinenprogramms einer 1-Byte-Subtraktion:**

1. Übertragbit C setzen
   768 38 SEC

2. Zwei Zahlen laden und subtrahieren
   769 A9 LDA 3C
   770 3C
   771 E9 SBC 23
   772 23

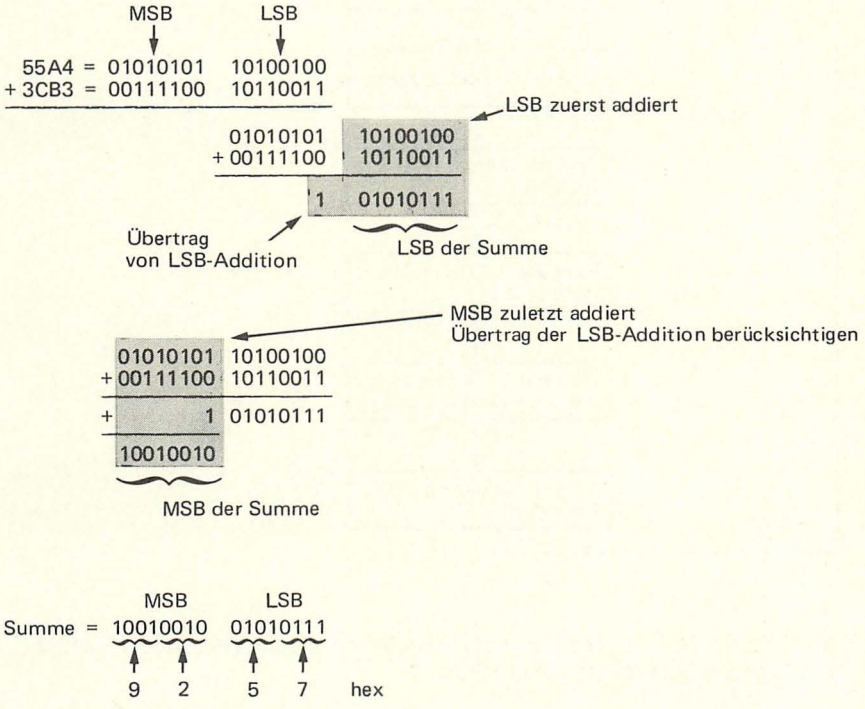
3. Differenz darstellen, Rückkehr zum BBS
   773 2Ø JSR FDDA
   774 DA
   775 FD
   776 6Ø RTS

Subtraktionen mit positiven und negativen Differenzen als Ergebnis werden wir im Zusammenhang mit 2-Byte-Subtraktionen behandeln.

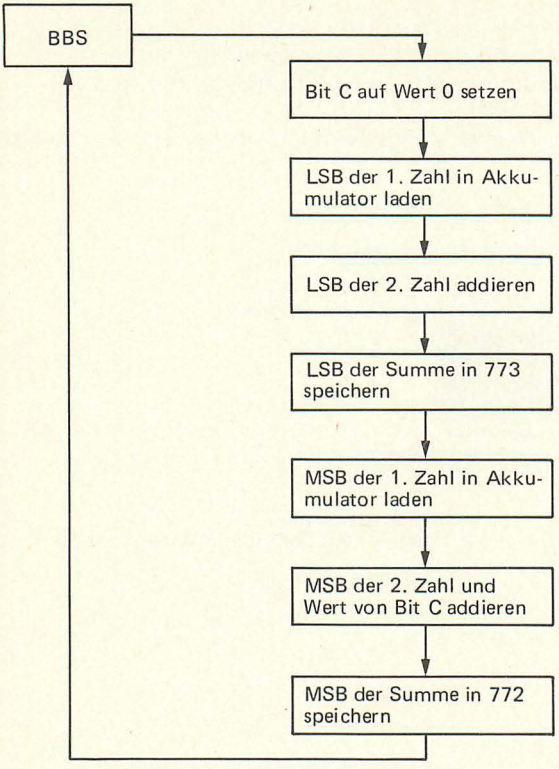
2-Byte-Addition

Das folgende Maschinenprogramm zeigt die Addition von 2 Zahlen, die jeweils durch 2 Bytes dargestellt sind. Als Sumanden sind jetzt Zahlen im Bereich 0 . . . 65535 dez möglich, und der Maximalwert der Summe kann 01FFFF hex oder 131071 dez erreichen.

An der folgenden Ausführung einer 2-Byte-Addition können wir die erforderlichen Programmschritte erkennen:



Aus diesem Ablauf wurde das folgende Flußdiagramm abgeleitet:



Zum Ablegen der 2 Bytes MSB, LSB für jeden der Sumanden und für die Summe wurden folgende Speicherstellen gewählt:

Speicheradressen dez	Speicherinhalte
790	LSB
791	MSB } des 1. Summanden
792	LSB
793	MSB } des 2. Summanden
794	MSB
795	LSB } der Summe

Zur Programmeingabe über das Betriebssystem BBS verwenden wir folgende ANGABEN ZUM MASCHINENPROGRAMM:

Anfangsadresse: 768
Anzahl der Bytes: 26

Hier die Codes des **Maschinenprogramms zur 2-Byte-Addition:**

1. Übertragbit C löschen

```
768 18      CLC
```

2. LSB laden, addieren; LSB-Summe speichern

```
769 AD      LDA 0316      LSB des 1. Summanden laden
770 16
771 Ø3

772 6D      ADC 0318      LSB des 2. Summanden addieren;
773 18                    bei Übertrag Bit C = 1 setzen
774 Ø3

775 8D      STA 031B      LSB-Teilsumme in 795 speichern
776 1B
777 Ø3
```

3. MSB laden, addieren; MSB-Summe speichern

```
778 AD      LDA 0317      MSB des 1. Summanden laden
779 17
780 Ø3

781 6D      ADC 0319      MSB des 2. Summanden und Übertragbit C addieren;
782 19                    bei neuem Übertrag Bit C = 1 setzen
783 Ø3

784 8D      STA 031A      MSB-Teilsumme in 794 speichern
785 1A
786 Ø3

787 6Ø      RTS           Rückkehr ins Hauptprogramm
```

4. Daten

```
788 ØØ      BRK  }
789 ØØ      BRK  }  "Platzhalter"

790 A4      LSB  }
791 55      MSB  }  1. Summand

792 B3      LSB  }
793 3C      MSB  }  2. Summand

794         MSB  }
795         LSB  }  Summe
```

Die einzelnen Programmschritte werden im folgenden kommentiert:

SCHRITT 1

Mit CLC wird das Übertragbit C im Statusregister auf den Wert 0 gesetzt. Bei Beginn der Additionen liegt also kein Übertrag vor.

SCHRITT 2

Dieser Programmabschnitt bildet eine 1. Teilsumme (LSB-Teilsumme) aus den niedrigstwertigen Bytes LSB beider Summanden.

Mit LDA 0316 wird das LSB des 1. Summanden von Speicherstelle 0316 hex (790 dez) in den Akkumulator A geladen.

Mit ADC 0318 werden der Inhalt der Speicherstelle 0318 hex (792 dez), d.h. das LSB des 2. Summanden, unter Berücksichtigung des Übertragbit C zum Inhalt des Akkumulators A addiert. Das Übertragbit C war in Schritt 1 auf den Wert 0 gesetzt

worden. Die entstehende Teilsumme steht im Akkumulator A. Führt diese Addition zu einem Übertrag, dann wird das Übertragbit C auf den Wert 1 gesetzt.

Mit STA 031B wird die LSB-Teilsumme vom Akkumulator A unter Speicheradresse 031B hex (795 dez) abgelegt.

SCHRITT 3

Dieser Programmabschnitt bildet die 2. Teilsumme (MSB-Teilsumme) aus den höchstwertigen Bytes, MSB, beider Summanden und einem Übertrag, der bei Bildung der LSB-Teilsumme entstanden war.

Mit LDA 0317 wird zunächst das MSB des 1. Summanden in den Akkumulator A geladen.

Mit ADC 0319 werden das MSB des 2. Summanden und der Inhalt des Übertragbit C zum Inhalt des Akkumulators A addiert. Der Wert des Übertragbit C wird zum niedrigstwertigen Bit des Akkumulatorinhalts addiert. Entsteht bei dieser MSB-Addition ein Übertrag, dann wird das Übertragbit C auf "1" gesetzt. In diesem Fall stellt der Inhalt des Übertragbit C das höchstwertige Bit der Summe dar und muß in die Summendarstellung aufgenommen werden.

Mit STA 031A wird die MSB-Teilsumme unter Adresse 794 dez abgelegt.

RTS schließt den Befehlsteil des Maschinenprogramms ab und löst die Rückkehr in das aufrufende Hauptprogramm aus.

SCHRITT 4

Die Speicheradressen 788 . . . 795 im Anschluß an das Maschinenprogramm dienen zur Datenspeicherung.

Die Speicheradressen 788, 789 enthalten für das vorliegende Maschinenprogramm keine Daten. Als "Platzhalter" wurde in diese Speicherstellen der Unterbrechungsbefehl BRK eingetragen. Führt der Mikroprozessor 6502 einen BRK-Befehl aus, dann springt er in eine Unterbrechungsroutine. Diese Routine unterbricht den Programmablauf und stellt auf dem Bildschirm die Programmzeile dar, in der die Unterbrechung erfolgte. Ein fälschliches Lesen dieses Teils der Datenliste führt also zu einer Fehlermeldung auf dem Bildschirm.

Die Summe dieser 2-Byte-Addition ist in 3 Speicherstellen abgelegt:

Bit 16:	im Übertragbit C
Bit 15 . . . 8 (MSB):	in Speicherstelle 794 dez
Bit 7 . . . 0 (LSB):	in Speicherstelle 795 dez

Die Wiedergabe von Bit 16 im Übertragbit C als höchstwertiger Teil der Summe haben wir bereits im 1-Byte-Additionsprogramm erläutert.

2-Byte-Subtraktion

Das Maschinenprogramm zur Subtraktion zweier 2-Byte-Zahlen hat den gleichen Aufbau wie das vorangegangene 2-Byte-Additionsprogramm.

Vor Beginn der Subtraktion wird das Übertragbit C mit dem Befehl SEC auf den Wert 1 gesetzt. Sollte die nachfolgende Subtraktion das Borgen einer "1" erforderlich machen, dann wird diese "1" aus dem Übertragbit C entnommen und dieses auf den Wert 0 gesetzt.

Der Befehl ADC im Additionsprogramm wird durch den Subtraktionsbefehl SBC ersetzt. Dieser Befehl adressiert ein Speicherwort, das vom Inhalt des Akkumulators subtrahiert wird; verläuft die Subtraktion mit einem negativen Übertrag, d.h. war das Borgen einer "1" erforderlich, dann setzt dieser Befehl das Übertragbit C auf den Wert 0.

Die ANGABEN ZUM MASCHINENPROGRAMM sind wie beim vorangegangenen Additionsprogramm:

Anfangsadresse: 768
Anzahl der Bytes: 26

Dieses Programm eignet sich für Subtrahenden im Wertbereich 0 . . . 65535. Das Programm liest die Subtrahenden aus den Speicherstellen 790 . . . 793 dez. In unserem Beispiel wird folgende Subtraktion ausgeführt:

dez	hex	bin
21924	55A4	0101 0101 1010 0100
− 15539	− 3CB3	− 0011 1100 1011 0011
6385	18F1	0001 1000 1111 0001

Die Differenz 18F1 hex steht in den Speicherstellen mit den Adressen 794, 795. Hier die Codes dieses **Maschinenprogramms zur 2-Byte-Subtraktion:**

1. Übertragbit C setzen

```
768 38    SEC
```

2. LSB laden, subtrahieren; LSB-Differenz speichern

```
769 AD    LDA 0316
770 16
771 Ø3

772 ED    SBC 0318
773 18
774 Ø3

775 8D    STA 031B
776 1B
777 Ø3
```

3. MSB laden, subtrahieren; MSB-Differenz speichern

```
778 AD    LDA 0317
779 17
780 Ø3

781 ED    SBC 0319
782 19
783 Ø3

784 8D    STA 031A
785 1A
786 Ø3

787 6Ø    RTS
```

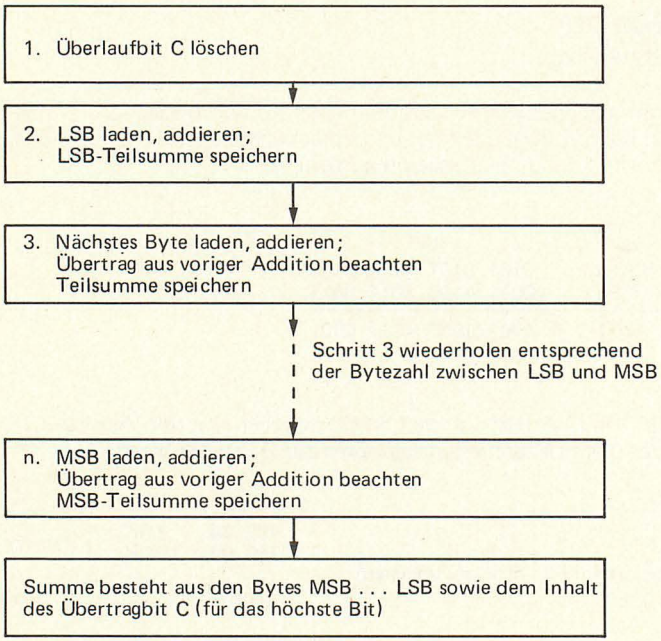
4. Daten

```
788 ØØ    BRK   } "Platzhalter"
789 ØØ    BRK   }

790 A4    LSB   } 1. Subtrahend
791 55    MSB   }

792 B3    LSB   } 2. Subtrahend
793 3C    MSB   }

794       MSB   } Differenz
795       LSB   }
```

Multi-Byte-Additionen

Auch Additionen von Zahlen, deren Darstellung mehr als zwei Bytes erfordern, verlaufen byteweise. Die Additionen beginnen mit dem niedrigstwertigen Byte, LSB, und enden mit der Addition des höchstwertigen Byte, MSB, wobei jeweils ein Übertrag aus der vorangegangenen Addition beachtet wird.

Hier der grundsätzliche Aufbau von Multi-Byte-Programmen:



1-Byte-Multiplikation

Der Mikroprozessor 6502 verfügt wie die meisten anderen Mikroprozessoren über keinen Befehl zur Multiplikation oder Division. Wir bilden daher die einzelnen Schritte einer Multiplikation oder Division mit den vorhandenen Maschinenbefehlen nach und fassen diese zu einem Programm zusammen.

Binärmultiplikationen sind äußerst einfach auf dem Mikroprozessor zu realisieren. Bei Dezimalzahlen sind es Ziffern aus dem Bereich 0 . . . 9, mit denen wir zu multiplizieren haben, bei Binärzahlen dagegen nur die Ziffern 0 und 1:

Vergleich dezimale/binäre Multiplikation (Beispiel):

dezimal:

250 x 0 = 0000
250 x 1 = 0250
250 x 2 = 0500
.
.
.
250 x 9 = 2250

} 10 mögliche Produkte

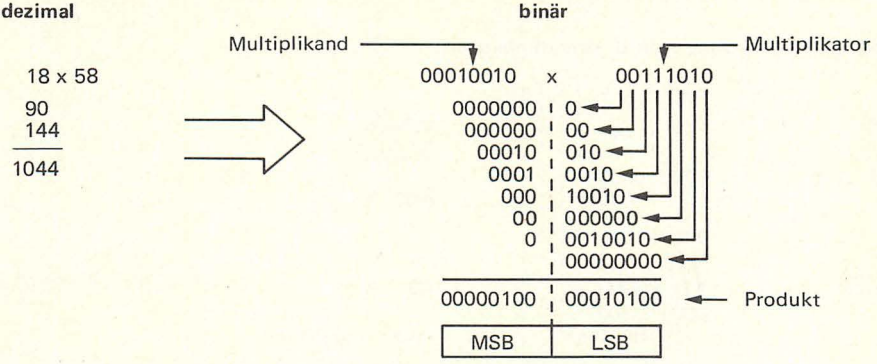
binär:

1010 x 0 = 0000
1010 x 1 = 1010

} 2 mögliche Produkte

Hier die Gegenüberstellung von dezimaler und binärer Multiplikation:

Vergleich dezimale/binäre Multiplikation (Beispiel):

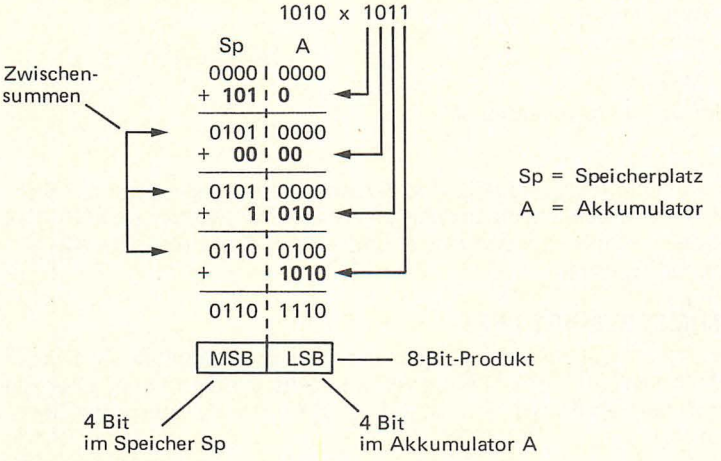


Aus dem Multiplikationsschema ersehen wir die wiederholt auszuführenden Schritte einer Binärmultiplikation:

VERSCHIEBEN UM 1 STELLE
BILDEN VON ZWISCHENSUMMEN

Für diese Schritte enthält das Repertoire des 6502 geeignete Maschinenbefehle. Hier ein 4-Bit-Beispiel, das die prinzipielle Ausführung von Binärmultiplikationen im Mikroprozessor leichter erkennen läßt:

Prinzipielle Ausführung einer Binärmultiplikation:

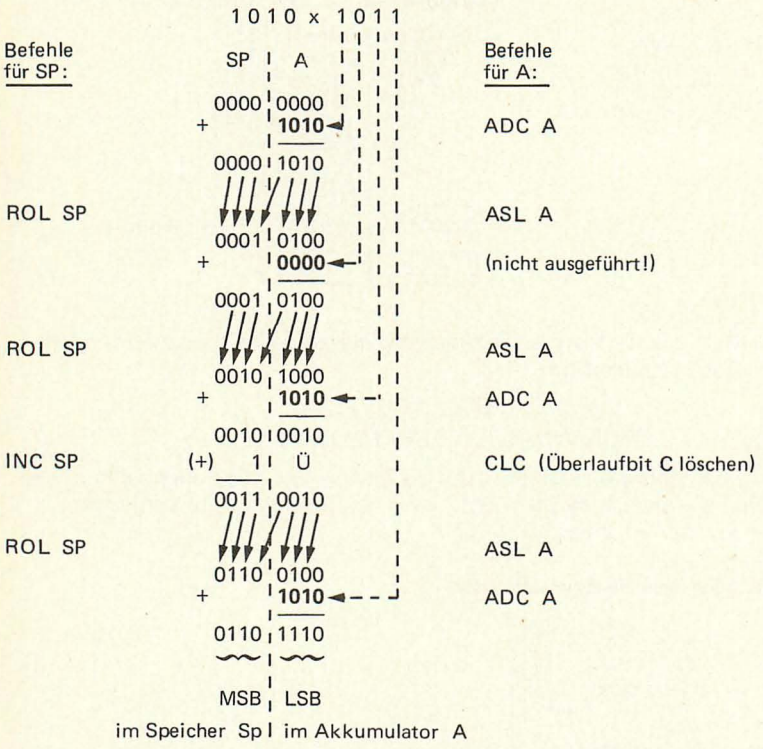


Nach diesem Schema schalten wir eine Speicherstelle Sp und den Akkumulator A zusammen, um das 8-Bit-Produkt von (4-Bit-Multiplikand) x (4-Bit-Multiplikator) aufzunehmen. Sie erkennen, wie der Multiplikand 1010 schrittweise verschoben wird und abhängig von den Ziffern 1,0 des Multiplikators zu den Speichern Sp, A

addiert oder nicht.

Wesentlich einfacher läßt sich eine Binärmultiplikation programmieren, wenn nicht der Multiplikand relativ zu Sp, A verschoben wird, sondern der Inhalt von Sp, A relativ zum Multiplikanden:

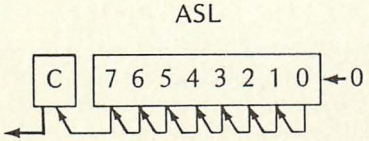
Tatsächliche Ausführung einer Binärmultiplikation:



Die Maschinenbefehle ROL und ASL sind Schiebebefehle, angewandt auf die Inhalte von Sp und A und anschließend ausführlich erklärt. Der Maschinenbefehl INC inkrementiert den Speicherinhalt um den Wert 1 und addiert damit einen im Akkumulator A entstandenen Übertrag.

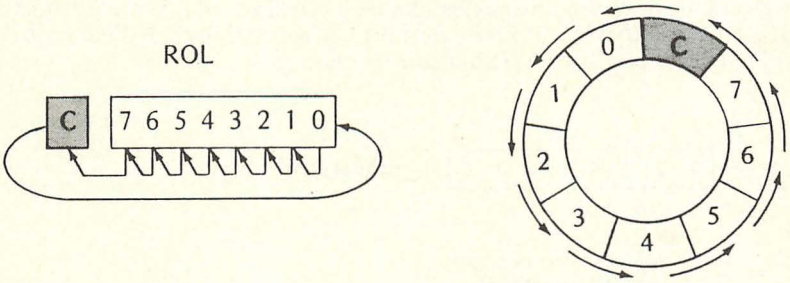
ASL ARITHMETIC SHIFT LEFT

Dieser Schiebebefehl verschiebt bei jeder Ausführung einen Speicherinhalt um 1 Stelle nach links, wobei das höchste Bit (Bit 7) nicht verloren geht, sondern in das Übertragbit C des Statusworts übertragen wird. ASL ist auf den Akkumulator und jeder Speicheradresse anwendbar.



ROL ROTATE (CONTENTS) OF (ACCUMULATOR) LEFT

Dieser Schiebebefehl rotiert bei jeder Ausführung den Akkumulatorinhalt über die Kette Akkumulator–Übertragbit–Akkumulator um 1 Stelle nach links.

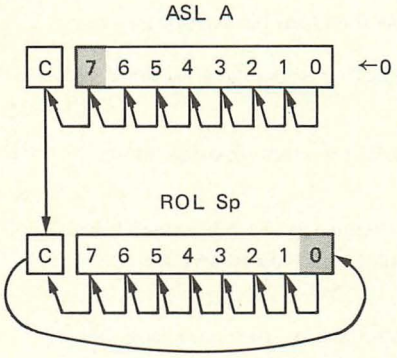


Hier die Anwendung dieser Befehle im Multiplikationsprogramm.

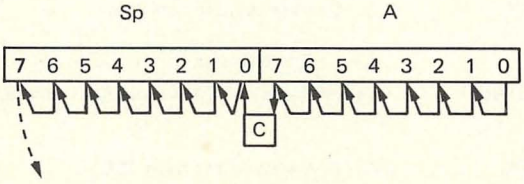
VERSCHIEBEN DER SPEICHERINHALTE VON Sp, A

Sp und A bilden in unserem Programm gedanklich einen 16-Bit-Speicher, dessen Inhalt um jeweils 1 Stelle nach links verschoben werden soll. Das aus der höchsten Stelle von Akkumulator A austretende Bit 7 soll in Bit 0 des anschließenden Speichers Sp gelangen. Die Befehlsfolge ASL A, ROL Sp löst diese Aufgabe.

Befehle:

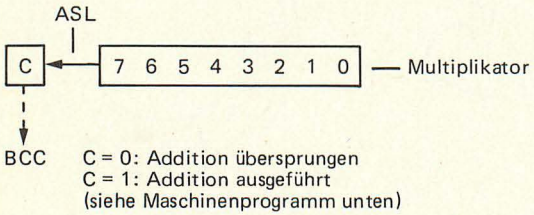


Wirkung:



LESEN DER BINÄRSTELLEN 0, 1 IM MULTIPLIKATOR

Die Binärstellen im Multiplikator bestimmen, ob der Multiplikand zum Akkumulator addiert wird oder nicht. Mit dem Befehl ASL wird der Speicher mit dem Multiplikator als Inhalt bitweise nach links verschoben. Im Übertragbit C erscheinen wie in einem Fenster der Reihe nach alle Binärstellen des Multiplikators. Mit dem Sprungbefehl BCC (BRANCH IF CARRY CLEAR) wird Bit C abgefragt. Für C = 0 wird die Addition des Multiplikanden zum Akkumulator übersprungen.



Hier die ANGABEN ZUM MASCHINENPROGRAMM bei Eingabe über das BBS:

Anfangsadresse: 768
Anzahl der Bytes: 35

Und hier die Codes des **Maschinenprogramms zur 1-Byte-Multiplikation:**

768 A2 769 Ø8	LDX 08	Schleifenzähler setzen
770 A9 771 ØØ	LDA 00	Akkumulator mit Wert 0 laden
772 8D 773 23 774 Ø3	STA 0323	MSB = 0 setzen (Speicher 803)
775 8D 776 24 777 Ø3	STA 0324	LSB = 0 setzen (Speicher 804)
778 ØA	ASL A	Akkumulator nach links verschieben
779 2E 780 23 781 Ø3	ROL 0323	MSB nach links verschieben
782 ØE 783 26 784 Ø3	ASL 0326	Multiplikator nach links verschieben
785 9Ø 786 Ø9	BCC 09	C = 0: Verzweigung nach 796
787 18	CLC	C = 1: Bit C löschen
788 6D 789 25 790 Ø3	ADC 0325	Multiplikand zum Akkumulator addieren
791 9Ø 792 Ø3	BCC 03	C = 0: Verzweigung nach 796

```
793 EE      INC 0323        C = 1: Übertrag auf MSB addieren
794 23
795 Ø3
796 CA      DEX             Zähler dekrementieren
797 DØ      BNE EB          Z = 0: Verzweigung nach 778
798 EB
799 8D      STA 0324        LSB des Produkts in 804 ablegen
800 24
801 Ø3
802 6Ø      RTS             Rücksprung
803         MSB  } 16-Bit-Produkt
804         LSB  }
805         8-Bit-Multiplikand
806         8-Bit-Multiplikator
```

Multiplikand, Multiplikator und Produkt stehen unter folgenden Adressen im Speicher:

Speicher-adressen dez	hex	Speicherinhalt
803	0323	MSB des Produkts
804	0324	LSB des Produkts
805	0325	Multiplikand
806	0326	Multiplikator

Nach Eingabe des Multiplikationsprogramms mit Hilfe des BBS können Sie es mit:

BSAVE MULT. OBJ. PR, A 768, L 35

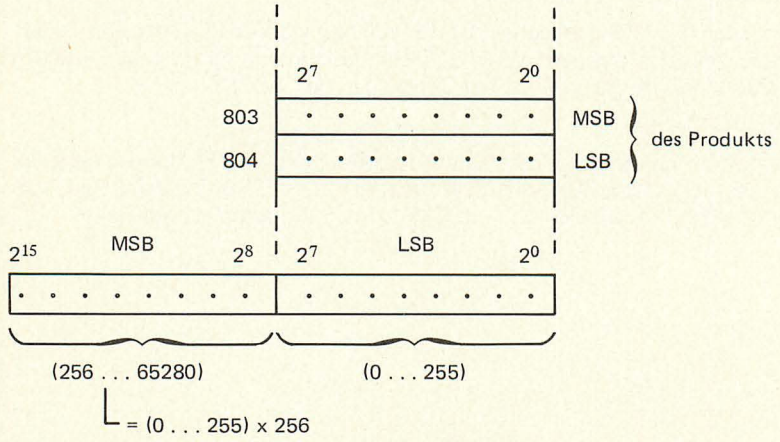
auf Diskette speichern. (Die Angabe OBJ. PR soll beim CATALOG des Diskettenin-halts sofort kenntlich machen, daß MULT ein Maschinenprogramm ist). Nach Rückholen des Maschinenprogramms von der Diskette in den APPLE mit:

BLOAD MULT

(falls dies erforderlich war), können Sie mit folgendem BASIC-Programm beliebige Multiplikanden und Multiplikatoren im Wertbereich 0 . . . 255 eingeben. Das Produkt wird als Dezimalzahl im Bereich 0 . . . 65535 auf dem Bildschirm dargestellt.

```
10  REM * 8-BIT-MULTIPLIKATION *
15  PRINT
20  PRINT "MULTIPLIKAND: "; SPC(
     2)
25  INPUT MD: POKE 805,MD
30  PRINT
35  PRINT "MULTIPLIKATOR:"; SPC(
     2)
40  INPUT MR: POKE 806,MR
45  CALL 768
50  PRINT
55  PRINT "PRODUKT:"; SPC( 2);
60  PRINT  PEEK (803) * 256 +  PEEK
     (804)
65  END
70  PRINT
75  PRINT "PRODUKT:"
80  PRINT  PEEK (803) * 256 +  PEEK
     (804)
90  END
```

Mit POKE 805, MD und POKE 806, MR werden der von Ihnen im Dialog erfragte Multiplikand MD und Multiplikator MR unter den oben genannten Adressen abgelegt. Mit PEEK (803) und PEEK (804) werden MSB und LSB des Produkts gelesen, entsprechend ihrer Stellenwertigkeit kombiniert.



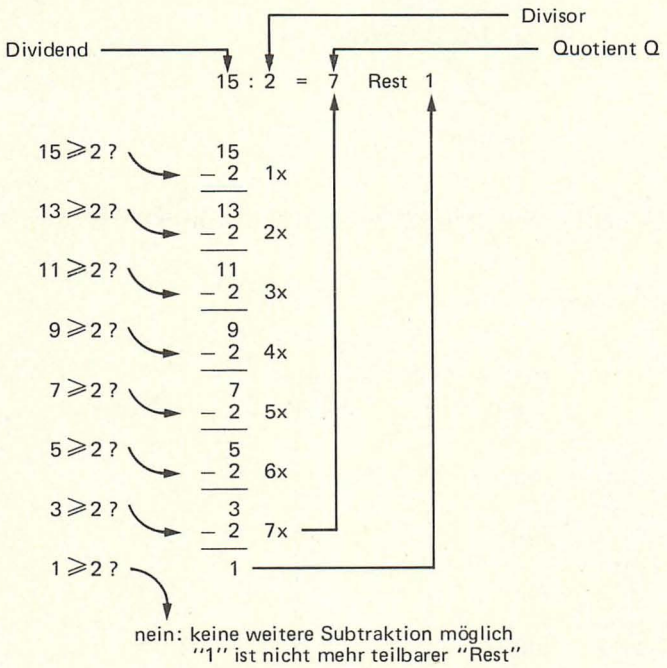
MSB + LSB: PRINT PEEK (803) * 256 + PEEK (804)

und mit PRINT auf dem Bildschirm dargestellt.

1-Byte-Division

Divisionen lassen sich auf wiederholte Subtraktionen zurückführen, wie folgendes Beispiel einer dezimalen Division zeigt:

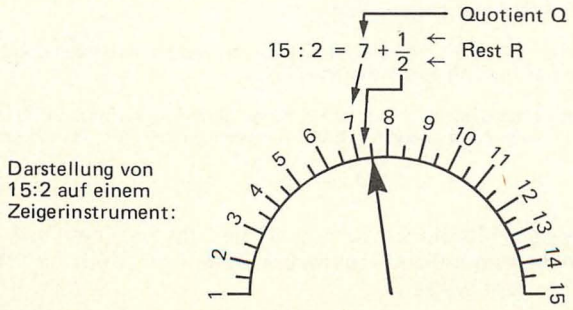
Rechenschema einer dezimalen Division:



Neu an der Division gegenüber einer Multiplikation ist das Auftreten eines nicht mehr teilbaren Rests neben dem Ergebnis, dem Quotienten.

Hier ein Beispiel für den Umgang mit dem Rest: Stellt der Dividend (im Beispiel oben: 15) eine Spannung dar, die nach Teilung (:2) auf einem Instrument anzuzeigen ist, dann würden Quotient und Rest wie folgt die Anzeige bilden:

Umgang mit dem "Rest" (Beispiel):



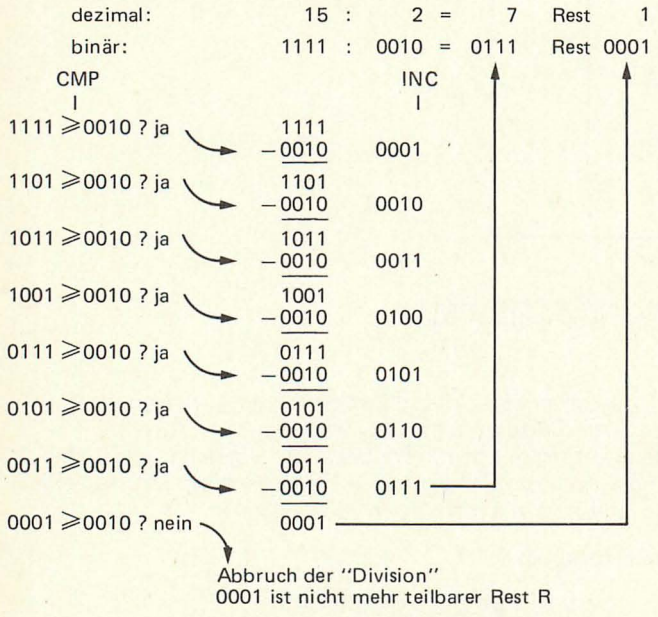
Allgemein besteht folgender Zusammenhang zwischen Divisor und möglichen Divisionsergebnissen:

Divisor	Mögliche Reste R	Mögliche Divisionsergebnisse
2	1	Q, Q + 1/2
3	1, 2	Q, Q + 1/3, Q + 2/3
4	1, 2, 3	Q, Q + 1/4, Q + 2/4, Q + 3/4
.	.	.
.	.	.
.	.	.

Q = Quotient

Übertragen auf Binärzahlen, sieht die oben gezeigte Dezimaldivision 15:7 wie folgt aus:

Rechenschema einer binären Division:



CMP: Die Subtraktion (Dividend–Divisor) wird solange fortgesetzt, wie die entstehenden Differenzen größer als der Divisor sind. Vergleichsbefehl CMP.

INC: Jede ausführbare Subtration wird gezählt, indem der Inhalt eines Quotientenspeichers um den Wert 1 inkrementiert wird. Anfangswert = 0000. Endwert = Quotient. Inkrementierbefehl INC.

Dieses 4-Bit-Rechenbeispiel läßt bereits prinzipiell die Schritte eines Divisionsprogramms erkennen, das jedoch im 8-Bit-Mikroprozessor des APPLE nur für folgende dezimale Wertebereiche geeignet wäre:

Dividend: 0 . . . 255 (1 Byte)
Divisor: 1 . . . 255 (1 Byte)
Quotient: 0 . . . 255 (1 Byte)

Nachfolgend geben wir jedoch ein leistungsfähiges Divisionsprogramm für folgende dezimale Wertebereiche wieder:

Dividend: 0 . . . 65025 (2 Byte)
Divisor: 1 . . . 255 (1 Byte)
Quotient: 0 . . . 255 (1 Byte)

Hier die ANGABEN ZUM MASCHINENPROGRAMM für das Betriebssystem BBS:

Anfangsadresse: 768
Anzahl der Bytes: 30

Der 2-Byte-Dividend wird als (MSB, LSB) eingegeben und steht mit Quotient, Rest und Divisor in folgenden Speicherstellen:

Speicheradresse dez	hex	Speicherinhalt
800	0320	Quotient *)
801	0321	Rest
802	0322	MSB des Dividenden
800	0320	LSB des Dividenden *)
803	0323	Divisor

*) am Ende der Division steht in Speicherstelle 803 der Quotient.

Schließlich die Codes des **Maschinenprogramms zur 1-Byte-Division:**

768	A2	LDX 08	Schleifenzähler X = 8 setzen
769	Ø8		
770	AD	LDA 0322	MSB des Dividenden in Akkumulator laden
771	22		
772	Ø3		
773	ØE	ASL 0320	LSB des Dividenden nach links verschieben
774	2Ø		
775	Ø3		
776	2A	ROL A	MSB des Dividenden nach links verschieben
777	BØ	BCS 06	C = 1: Verzweigen nach 785 (+ 6 Schritte)
778	Ø6		
779	38	SEC	C = 1 setzen
780	CD	CMP 0323	Vergleich (A – Divisor) $<$0 ?
781	23		
782	Ø3		
783	9Ø	BCC 06	C = 0: Verzweigen nach 791 (+ 6 Schritte)
784	Ø6		
785	ED	SBC 0323	Subtraktion (A – Divisor)
786	23		
787	Ø3		
788	EE	INC 0320	Quotientenspeicher 800 inkrementieren
789	2Ø		
790	Ø3		

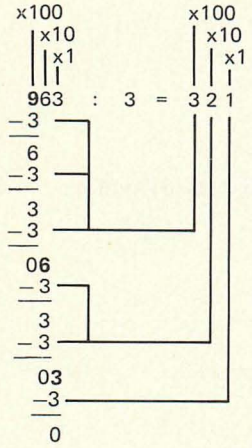
Fortsetzung →

```
791 CA    DEX          Schleifenzähler X dekrementieren
792 DØ    BNE EB       X≠0: Verzweigen nach 773 (– 21 Schritte)
793 EB
794 8D    STA 0321     Rest abspeichern nach 801
795 21
796 Ø3
797 6Ø    RTS          Rückkehr ins aufrufende Programm
800       Quotient; LSB des Dividenden
801       Rest
802       MSB des Dividenden
803       Divisor
```

Wichtig bei Benutzung dieses Programms ist, keine Quotienten größer als 255 entstehen zu lassen.

Wir werden im folgenden den Aufbau des Divisionsprogramms erläutern und ein einfaches BASIC-Programm angeben, mit dem die Funktion des Divisionsprogramms überprüft werden kann.

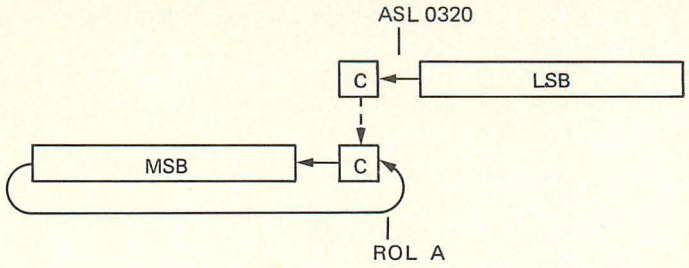
Dem Divisionsprogramm liegt die Alternative einer Divisionsausführung zugrunde, wie sie folgendes dezimale Schema zeigt:



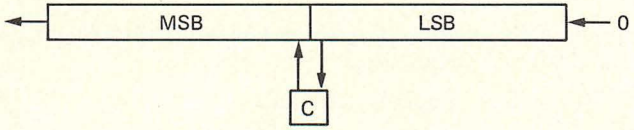
Die Division beginnt mit der höchstwertigen Dezimalstelle – im Divisionsprogramm mit dem höchstwertigen Byte MSB. Für Divisionen durch wiederholte Subtraktionen wird daher das MSB des Dividenden in den Akkumulator gebracht, das LSB bleibt in seiner Speicherstelle (hier: 800 dez).

Die erforderlichen Linksverschiebungen des 16-Bit-Dividenden führt wieder die Befehlsfolge ASL, ROL aus:

Befehle:



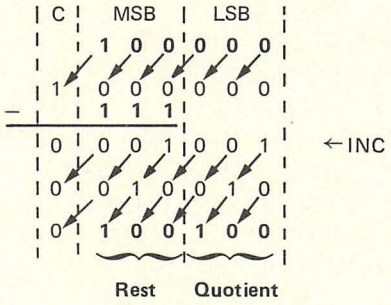
Wirkung:



Hier ein 6-Bit-Beispiel für die Arbeitsweise des Divisionsprogramms:

Division **dezimal:** 32 : 7 = 4 Rest 4

binär: 100000 : 111 = 100 Rest 100



Die Subtraktion des Divisors wird ausgeführt,

– wenn im höchstwertigen Bit des Dividenden eine "1" stand, die bei Linksverschiebung in das Übertragbit C gelangte:

```
777  BØ     BCS 06
778  Ø6
 .
 .
 .
785  ED     SBC 0323
786  23
787  Ø3
```

– wenn der Wert des MSB größer oder gleich dem Wert des Divisors ist:

```
780  CD     CMP 0323
781  23
782  Ø3
783  9Ø     BCC 06
784  Ø6
785  ED     SBC 0323
786  23
787  Ø3
```

Zur Überprüfung des Divisionsprogramms hier ein BASIC-Programm, das Sie wie folgt nach Dividend und Divisor fragt:

```
DIVIDEND  (Ø . . . 65Ø25):    ? □
DIVISOR   (1 . . . 255):      ? □
```

```
10  REM * 8-BIT-DIVISION *
20  HOME
30  PRINT "DIVIDEND (0...65025) :
     "; SPC( 2)
40  INPUT DD: PRINT
50  PRINT "DIVISOR  (1...255)    :
     "; SPC( 2)
60  INPUT DR: PRINT
70 A =  INT (DD / 4096)
80 X = DD - A * 4096
90 B =  INT (X / 256)
100 Y = X - B * 256
110 C =  INT (Y / 16)
120 D = Y - C * 16
130 MSB = A * 16 + B
140 LSB = C * 16 + D
150  POKE 802,MSB
160  POKE 800,LSB
170  POKE 803,DR
200  CALL 768
210  PRINT "QUOTIENT :"; SPC( 2)
220  PRINT  PEEK (800): PRINT
230  PRINT "REST     :"; SPC( 2)
240  PRINT  PEEK (801)
250  END
```

Ihre Eingabe des Dividenden wird zunächst in die Hexadezimalstellen A, B, C, D verwandelt (Zeilen 70 . . . 120). MSB und LSB des Dividenden lauten dann hexadezimal: MSB = A B, LSB = X D. (Hier sind A, B, C, D nur BASIC-Variable, keine Hexadezimalzahlen!). Schließlich werden MSB und LSB als Dezimalzahlen ausgedruckt (Zeilen 130, 140):

```
MSB = A * 16 + B
LSB = C * 16 + D
```

und mit POKE-Befehlen an die bereits oben genannten Speicherstellen gebracht. Mit CALL 768 (Programmzeile 200) ruft das BASIC-Programm unser Divisionsprogramm auf und liest mit PEEK-Befehlen (Zeilen 210 . . . 240) Quotient und Rest aus den Speicherstellen des Maschinenprogramms, um sie wie folgt dezimal darzustellen:

```
DIVIDEND   (Ø . . . 65Ø25):    ? 65Ø25
DIVISOR    (1 . . . 255):      ? 255
QUOTIENT:  255
REST:      Ø
```

Dieses BASIC-Programm dient selbstverständlich nur zum Überprüfen des Maschinenprogramms und nicht zum Ersetzen des Divisionsbefehls in BASIC. Unser Divisionsprogramm arbeitet jedoch erheblich schneller als der entsprechende BASIC-Befehl: es benötigt zwischen 150 und 230 Mikrosekunden.

10
Alternativen zum BASIC BBS

Einführung

In den vorangegangenen Kapiteln konnten wir mit BASIC arbeiten, um Maschinenprogramme in den APPLE einzugeben und aufzurufen. Wir hatten uns hierfür ein "BASIC-Betriebssystem" – das BBS – geschrieben, das wir zu Experimenten mit den Maschinenprogrammen leicht um geeignete BASIC-Befehle erweitern konnten.

Wir werden jetzt professionellere Möglichkeiten der Eingabe und Überprüfung von Maschinenprogrammen kennenlernen, mit denen jeder APPLE-Computer ausgestattet ist, und zwar in Form seiner fest eingespeicherten Programme

- APPLE SYSTEM MONITOR
  und
- APPLE MINI-ASSEMBLER

Der APPLE SYSTEM MONITOR

Der System-Monitor besteht aus einer Reihe fest im APPLE eingespeicherter Unterprogramme, die Sie ganz einfach durch Tasteneingaben aktivieren können und die Ihnen in Dialogform erlauben:

- Speicherinhalte zu ändern
- Maschinenprogramme einzugeben
- Maschinenprogramme zu starten
- Registerinhalte zu inspizieren
- Registerinhalt zu ändern

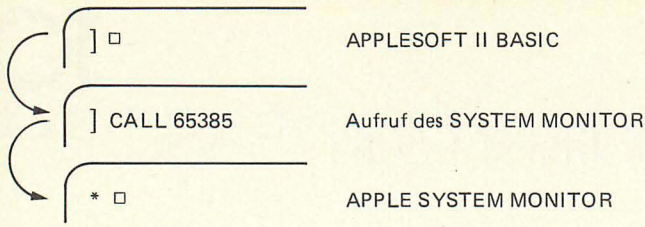
(und weitere Operationen)

AUFRUF DES SYSTEM MONITOR

Ihr Benutzerhandbuch mag noch andere Möglichkeiten nennen, in den SYSTEM MONITOR zu gelangen – auf jeden Fall gelingt es mit dem Aufruf

```
CALL 65385
```

(Der SYSTEM MONITOR beginnt mit Speicheradresse 65385 dez oder FF69 hex). Als Bestätigung erscheint als Hinweiszeichen ein Stern (*):

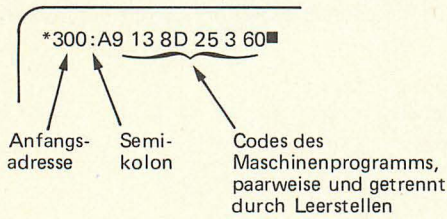


PROGRAMMEINGABE

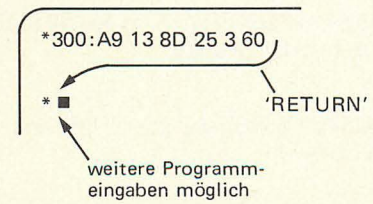
Hier ein kurzes Maschinenprogramm, an dem wir zeigen wollen, wie einfach Maschinenprogrammeingaben mit dem SYSTEM MONITOR sind:

Speicheradressen		**Maschinenbefehle**	
dez	hex	hex	mnem
768	Ø3ØØ	A9	LDA 13
769	Ø3Ø1	13	
770	Ø3Ø2	8D	STA 0325
771	Ø3Ø3	25	
772	Ø3Ø4	Ø3	
773	Ø3Ø5	6Ø	RTS

Anders als beim BBS werden Speicheradressen hexadezimal eingegeben. Wir beginnen mit 300 (eine führende Null **0**300 muß nicht angegeben werden), dem ein Doppelpunkt : und die hexadezimalen Codes unseres Maschinenprogramms paarweise folgen – jeweils getrennt durch eine Leerstelle:



Bis zu 255 Zeichen können je Zeile eingegeben werden. Wir schließen die Eingaben mit 'RETURN' ab:



und haben bereits das erste Maschinenprogramm über den SYSTEM MONITOR eingegeben!

PROGRAMMAUFLISTUNG

Unsere 6 Eingaben stehen jetzt in den Adressen 300 . . . 305. Der SYSTEM MONITOR listet uns die Speicherinhalte nach folgender Aufforderung auf:

```
*300:A9 13 8D 25 3 60 <--- Ihre Programmeingaben

*300.305■ <--------------- Programmauflistung durch Eingabe von
                           Anfangsadresse. Endadresse
```

Hier die Auflistung:

```
*300:A9 13 8D 25 3 60 <--- Ihre Programmeingaben

*300.305 <---------------- Programmauflistung angefordert

0300– A9 13 8D 25 03 60 <- Auflistung
*■
```

PROGRAMMSTART

Und nun der Start des eben abgespeicherten Maschinenprogramms. (Seine Aufgabe: den Zahlenwert 13 hex unter Speicheradresse 0325 hex oder 805 dez abzulegen).

```
*300:A9 13 8D 25 3 60 <--- Programmeingaben

*300.305 <---------------- Programmauflistung angefordert

0300–A9 13 8D 25 03 60 <-- Auflistung
*300G■ <------------------ Programmstart angefordert

*■ <---------------------- 300 Anfangsadresse des zu startenden Programms
                           G für GO
```

Das Programm ist abgelaufen, der SYSTEM MONITOR erwartet mit *■ weitere Eingaben. Wir werden uns das Ergebnis des Programmlaufs ansehen.

PROGRAMMERGEBNIS DARSTELLEN

Als Programmergebnis erwarten wir den Wert 13 hex in Speicherstelle 0325 hex. Hier die Form, um diese Speicherstelle mit dem SYSTEM MONITOR zu inspizieren:

```
*300:A9 13 8D 25 3 60

*300.305

0300–A9 13 8D 25 03 60
*300G

*325          <-- Inhalt der Speicheradresse 0325 inspizieren

0325– 13      <-- Speicherinhalt: 13 hex = 19 dez
*■
```

PROGRAMMVERSCHIEBUNG IM SPEICHER

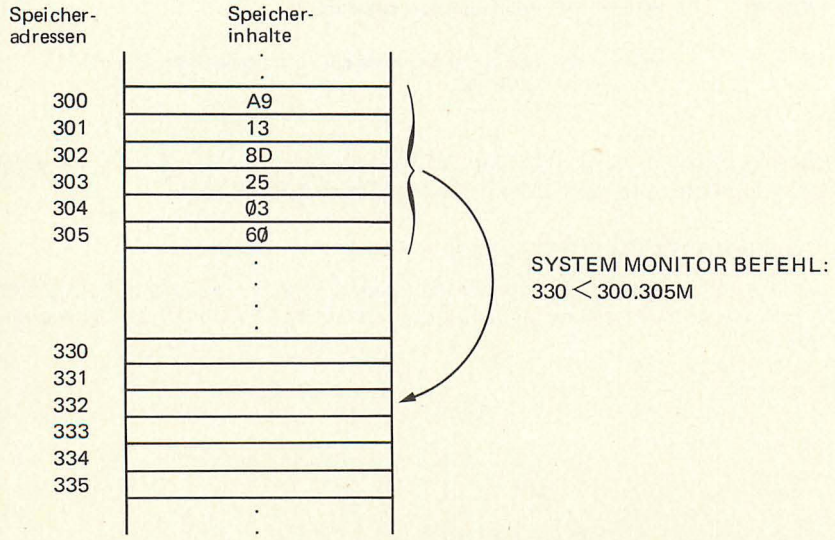
Bisher haben wir bei allen Maschinenprogrammen als Anfangsadresse 768 dez (0300 hex) gewählt. Wollen wir mit mehreren Maschinenprogrammen gleichzeitig arbeiten, dann dürfen sie sich nicht im Speicher überlappen.

Hier der Befehl im SYSTEM MONITOR, der Maschinenprogramme im Speicher verschiebt:

ziel < anfang . ende M

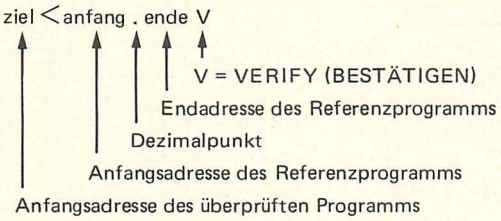
- M = MOVE (VERSCHIEBE)
- Endadresse des Programmoriginals
- Dezimalpunkt
- Anfangsadresse des Programmoriginals
- Anfangsadresse der Programmkopie

und hier eine Anwendung:



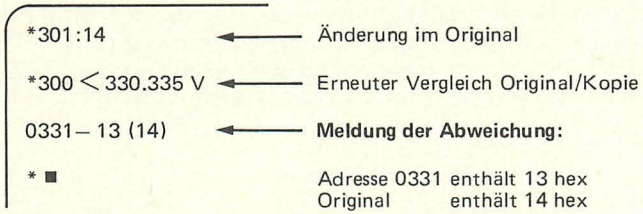
VERGLEICH VON SPEICHERBEREICHEN

Wollen wir überprüfen, ob die Speichereintragungen für Programmoriginal und Programmkopie übereinstimmen, dann lautet der Befehl des SYSTEM MONITORS:



Hier ein Beispiel für die Ausführung bei Übereinstimmung von Original und Kopie:

Ändern wir im Originalbereich eine der Eintragungen und vergleichen dann beide Speicherbereiche, so erhalten wir folgende Meldung:



PROGRAMMÄNDERUNGEN

Einer Programmänderung können wir wie im BBS die Auflistung des Programms vorangehen lassen. Der SYSTEM MONITOR läßt uns jedoch den Adreßbereich angeben, der uns dargestellt werden soll:

```
* 03C0.03E7                    ← Wahl des Adreßbereichs
03C0 – 2D  2E  2F  30  31  32  FF  FF
03C8 – 33  34  35  36  37  38  FF  39
03D0 – 4C  BF  9D  4C  84  9D  4C  FD   } Auflistung
03D8 – AA  4C  B5  B7  AD  0F  9D  AC
03E0 – 0E  9D  60  AD  C2  AA  AC  C1
```

Jede Bildschirmzeile stellt 8 Speicherinhalte dar, deren Adressen sich leicht aus der Anfangsadresse am Zeilenanfang erkennen lassen, z.B.

```
Ø3CØ . . . Ø3C7
Ø3C8 . . . Ø3CF
```

Wir haben bereits kennengelernt, wie Speicherinhalte zu ändern sind. Beispielsweise für Adresse 03C3, deren Inhalt von 3Ø in 4Ø geändert werden soll, wie folgt:

```
* 3C3 : 4Ø              ('RETURN')
```

Erneute Auflistung dieser Zeile zeigt, daß die Änderung ausgeführt wurde:

```
* 3CØ.3C7              ('RETURN')
03CØ – 2D  2E  EF  4Ø  31  32  FF  FF
                   ↑
```

REGISTERINHALTE INSPIZIEREN/ÄNDERN

Der Mikroprozessor 6502 im APPLE verfügt über folgende Daten- und Anzeigeregister:

A	Akkumulator
X	Indexregister X
Y	Indexregister Y
P	Statusanzeige
S	Zeiger zum Stapelspeicher

Aus Registerinhalten können wir die Arbeitsschritte des Mikroprozessors beim Programmablauf rekonstruieren – um beispielsweise Programmfehler zu suchen.

Der SYSTEM MONITOR stellt alle Registerinhalte nach Eingabe von CTRL E und 'RETURN' auf dem Bildschirm dar, beispielsweise folgende:

```
*                    ← CTRL E und 'RETURN' eingeben

A=FF X=FF Y=FF P=00 S=FF   ← Registerinhalte angezeigt
* ■
```

Die Registerinhalte lassen sich ebenso leicht ändern: nach dem Hinweiszeichen (*) geben wir ein Semikolon und die neuen Registerinhalte ein:

```
A=FF X=FF Y=FF P=00 S=FF   ← Ursprüngliche Registerinhalte
*:88 FF 80 33 40           ← Eingabe von Änderungen

*                          ← CTRL E und 'RETURN'

A=88 X=FF Y=80 P=33 S=40   ← Geänderte Registerinhalte
* ■
```

In diesem Beispiel haben sich die einzelnen Bits N, V, B, D, I, Z, C der Statusanzeige P wie folgt geändert:

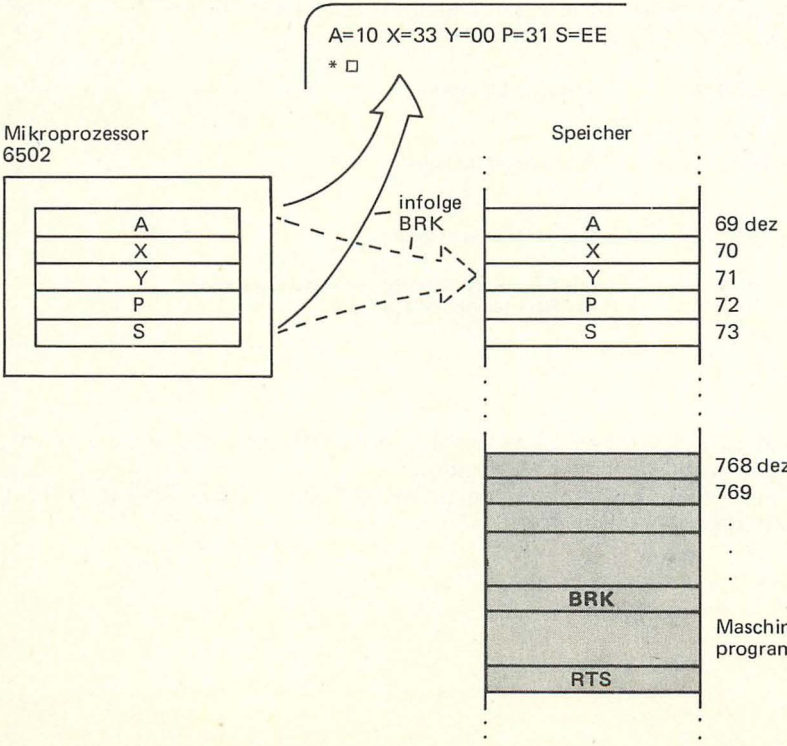
```
                NVBDIZ
P = 00 :        00000000
```

In diesem Beispiel haben sich die einzelnen Bits N, V, B, D, I, Z, C der Statusanzeige P wie folgt geändert:

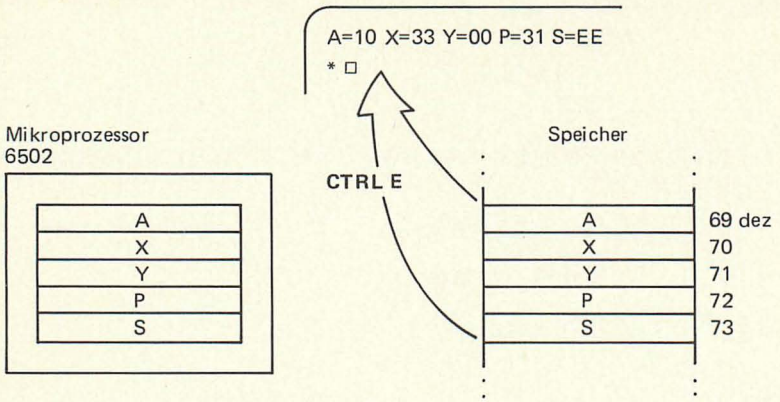
```
                N V  B D I Z C
P = 00 :        0 0 0 0 0 0 0 0
P = 33 :        0 0 0 1 0 0 0 1
```

Zu beachten:
Mit CTR L E stellt der SYSTEM MONITOR nur Registerinhalte dar, die zuvor im Speicher (ab Adresse 69 dez, 45 hex) abgelegt worden waren. Anlässe, zu denen Registerinhalte in den Speicher gerettet werden, sind der Aufruf des MONITORS mit CALL 65385 oder die Ausführung des Unterbrechungsbefehls BRK durch ein Maschinenprogramm.

Registerinhalte dargestellt nach BRK:



Registerinhalte dargestellt durch CTRL E:



Vor Ablauf eines Maschinenprogramms überträgt der SYSTEM MONITOR die abgespeicherten Registerinhalte in den Mikroprozessor, wo sie als Anfangswerte dienen. Die oben besprochene Möglichkeit, Registerinhalte zu ändern, bezieht sich nur auf die abgespeicherten – nicht die tatsächlichen – Registerinhalte.

Wichtig zu wissen ist auch, daß mit Ausführung eines BRK-Befehls sämtliche momentanen Registerinhalte auf dem Bildschirm dargestellt werden. Zur Probe geben wir folgendes Subtraktionsprogramm ein:

```
300 A9    LDA 22      Minuend 22 laden
301 22
302 A8    TAY
303 A9    LDA 33      Subtrahend 33 laden
304 33
305 AA    TAX
306 E9    SBC 22      Subtraktion ausführen
307 22
308 ØØ    BRK         mit BRK Programmabbruch und Darstellung
                      der Registerinhalte auslösen
```

(Subtrahend 33 und Minuend 22 werden in X- und Y-Register geladen, um sie anschließend als Registerinhalte darzustellen).

Vor Aufruf des Subtraktionsprogramms mit *300G ('RETURN') ändern wir jeweils die Registerinhalte wie folgt:

```
                                                    C=0
                                                     |
*:ØØ ØØ ØØ ØØ ØØ                          P: 00000000
A=ØØ X=ØØ Y=ØØ P=ØØ S=ØØ
*300G ←Start des Maschinenprogramms
0307 –  A=10 X=33 Y=22 P=31 S=EC
```

	HEX
	33
	–22
	11
C=0:	– 1
	10

```
                                                    C=1
                                                     |
*:ØØ ØØ ØØ Ø1 ØØ                          P: 00000001
A=ØØ X=ØØ Y=ØØ P=Ø1 S=ØØ
*300G ← Start des Maschinenprogramms
0307 –  A=11 X=33 Y=22 P=31 S=EA
```

	HEX
	33
	–22
	11
C=1:	– 0
	11

Dieser Versuch zeigt, daß der Subtraktionsbefehl SBC den Inhalt des Übertragbits C berücksichtigt und seine Wirkung wie folgt dargestellt werden kann:

SBC:
(AKKUMULATOR – MINUEND) – 1 + C

VERLASSEN DES SYSTEM MONITOR

Zur Rückkehr in die Programmiersprache, von der aus Sie den SYSTEM MONITOR aufgerufen hatten – gewöhnlich also APPLESOFT BASIC – geben Sie folgende Kommandos ein:

```
]■                    APPLESOFT BASIC                    ]■
 ↓                                                        ↑
 CALL 65385                                          CTRL C
                                                     oder
                                                     CTRL B
*■                    SYSTEM MONITOR                  *■
```

Der APPLE MINI-ASSEMBLER

Assembler sind Hilfsprogramme, mit denen wir Maschinenprogramm mnemonisch eingeben können, also beispielsweise in folgender Form:

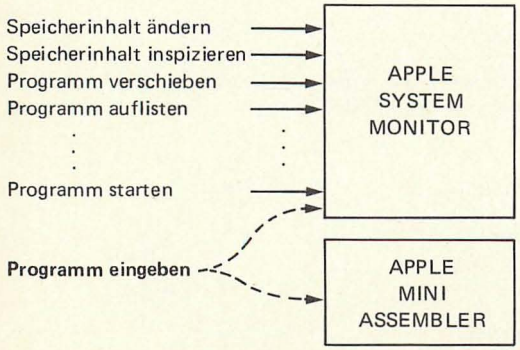
```
0300  SED
      CLC
      LDA 23
      ADC 18
      JSR FDDA
      RTS
```

Lediglich die Anfangsadresse des Speicherbereichs, in dem das Maschinenprogramm stehen soll, ist anzugeben. Der Assembler erkennt, wieviele Bytes jeder der Maschinenbefehle im Speicher belegt, erhöht entsprechend die Anfangsadresse und legt das Maschinenprogramm im Speicher ab.

Der APPLE MINI-ASSEMBLER heißt "mini", da er eine der Möglichkeiten vollständiger Assembler nicht bietet: Adreß- und Operandenangaben in symbolischer Form. Hier ein Beispiel für symbolische Adressen und Operanden:

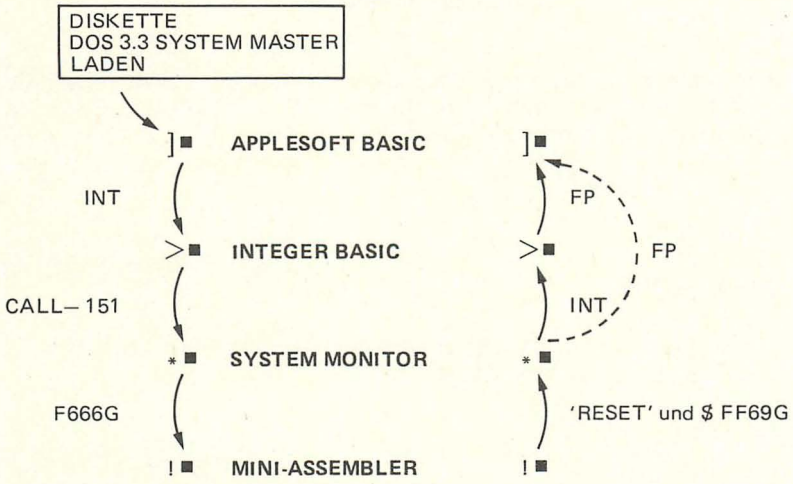
```
symbolische                                    symbolische
Adressen                                       Operanden
                LDY  09
LOOP            LDA  DATA, Y
                JSR  UNTPGR
                DEY                            Schleife LOOP
                BPL  LOOP
                RTS
DATA            0301
                A713
                 :                             Datenliste DATA
                 :
UNTPGR          CLC
                 :                             Unterprogramm UNTPGR
                 :
```

Man kann den APPLE MINI-ASSEMBLER als bequeme Alternative zum SYSTEM MONITOR betrachten – aber nur für die Funktion "Maschinenprogrammeingabe". Der MINI-ASSEMBLER hat keine weitere Funktion!



AUFRUF DES MINI-ASSEMBLER

Der MINI-ASSEMBLER ist nur über INTEGER BASIC zu erreichen. Haben Sie die APPLESOFT II ROM-Karte in Ihrem Computer, dann stellen Sie auf der Hinterseite Ihres Computers den Schalter auf INTEGER BASIC. Für APPLE IIe gilt beispielsweise folgender Weg:



Der MINI-ASSEMBLER ist erreicht, wenn auf dem Bildschirm ein Ausrufezeichen erscheint.

PROGRAMMEINGABE

Die Eingabe von Maschinenbefehlen im MINI-ASSEMBLER hat folgende Gestalt:

```
        Symbol für
        "Mini-Assembler"
              |
        !  ADRESSE  :  BEFEHL

z.B.    !  300      :  SED
              |
        Adresse      Maschinenbefehl
        hexadezimal  mnemonisch
```

Nach Quittieren der Eingabe mit 'RETURN' erscheint folgende Rückmeldung:

```
1. EINGABE        : ! 300 : SED

   RÜCKMELDUNG :  0300–          F8          SED
                    |            |           |
                  Adresse      Code        Mnemonik
                               des Maschinenbefehls
```

Die erste Programmzeile liegt vor. Für alle weiteren Programmzeilen müssen Sie keine Speicheradressen angeben – dies übernimmt der MINI-ASSEMBLER. Sie geben von jetzt ab nur noch die Maschinenbefehle ein:

```
                  Leertaste!
                     ↓
2. EINGABE        : ! CLC

   RÜCKMELDUNG :  0301–          18          CLC
```

Wichtig ist das Leerzeichen unmittelbar nach dem !-Zeichen; hier die Fehlermeldung, wenn die Leertaste vergessen wurde:

```
FALSCHE
EINGABE           : !CLC

RÜCKMELDUNG :     ! CLC       Leerzeichen zwischen ! und CLC fehlte.

                  ! □         erneute Eingabe erwartet
```

Als Beispiel geben wir folgendes Maschinenprogramm ein:

```
300  SED
     CLC
     LDA #23
     ADC #18
     JSR $FDDA
     RTS
```

Am Ende der Eingabe im MINI-ASSEMBLER haben wir folgende Bildschirmdarstellung:

```
0300–    F8          SED
0301–    18          CLC
0302–    A9  23      LDA  #$23
0304–    69  18      ADC  #$18
0306–    20  DA FD   JSR  $FDDA
0309–    60          RTS
```

Die vom Programmierer gewünschte Adressierungsart eines z.B. LDA-Befehls ersieht der MINI-ASSEMBLER aus der Größe des Operanden bzw. aus der Schreibweise:

Ihre Eingabe (Beispiele)

vom MINI-ASSEMBLER erkannter Code

```
! LDA #13
0300-        A9   13          LDA #$13

! LDA 13
0301-        A5   13          LDA $13

! LDA 13, X
0302-        B5   13          LDA $13, X

! LDA 03FF
0303-        AD   FF  03      LDA $03FF

! LDA 03FF, X
0304-        BD   FF  03      LDA $03FF, X

! LDA (13, X)
0305-        A1   13          LDA ($13, X)

! LDA (13), Y
0306-        B1   13          LDA ($13), Y
```

Nach Eingabe des Maschinenprogramms mit Hilfe des MINI-ASSEMBLERS kehren wir zum SYSTEM MONITOR zurück, um uns beispielsweise das Programm auflisten zu lassen:

```
! ■   MINI-ASSEMBLER
  'RESET' und
* ■   SYSTEM MONITOR
```

```
* 300L                    ← Auflistung des Maschinenprogramms veranlassen

0300- F8          SED
0301- 18          CLC
0302- A9 23       LDA #$23
0304- 69 18       ADC #$18
0306- 20 DA FD    JSR $FDDA
0309- 60          RTS
030A- FF          ???
030B- FF          ???
030C- FF          ???
030D- FF          ???
030E- FF          ???
030F- FF          ???
0310- FF          ???
0311- FF          ???
0312- FF          ???
0313- FF          ???
0314- FF          ???
0315- FF          ???
0316- FF          ???
0317- FF          ???
*
```

aufgelistetes Maschinenprogramm

ohne Bedeutung für uns

11

Was tun mit den Maschinenprogrammen?

Sämtliche Maschinenprogramme in diesem Buch können Sie über unser Betriebssystem BBS, den APPLE SYSTEM MONITOR oder den APPLE MINI-ASSEMBLER eingeben. Auch wenn Sie zwischen den verschiedenen Betriebssystemen und Programmiersprachen wechseln, oder mit dem BASIC-Befehl NEW alle BASIC-Programme im Speicher gelöscht haben – das Maschinenprogramm bleibt im Speicher stehen (solange Sie den Computer nicht ausschalten).

Wir werden über folgende Möglichkeiten sprechen, die wir mit einem Maschinenprogramm im Speicher des APPLE haben:

- Abspeichern auf Diskette
- Laden von Diskette
- Aufrufen durch BASIC-Programme
- Weitere Bearbeitung

Abspeichern auf Diskette

Ein Maschinenprogramm sei über das Betriebssystem BBS eingegeben und auch gestartet worden. Nach Ablauf des Maschinenprogramms endet auch das Betriebssystem BBS.

Das Maschinenprogramm steht jetzt im Speicher des APPLE-Computers und kann durch Eingabe des folgenden BASIC-Befehls auf einer Diskette abgespeichert werden:

BSAVE name, **A** anfangsadresse, **L** länge

Dem Maschinenprogramm muß ein name gegeben werden, aus dem möglichst ersichtlich sein sollte, daß es sich um ein Maschinenprogramm handelt. Weiterhin sind anfangsadresse A und programmlänge L als Dezimalzahlen anzugeben. Beide Angaben wurden bereits zu Beginn des Betriebssystems BBS im Dialog erfragt.

Den Befehl BSAVE geben Sie nach Rückkehr zu INTEGER BASIC (< ■) oder APPLESOFT BASIC (] ■) über das Tastenfeld ein.

Laden von Diskette

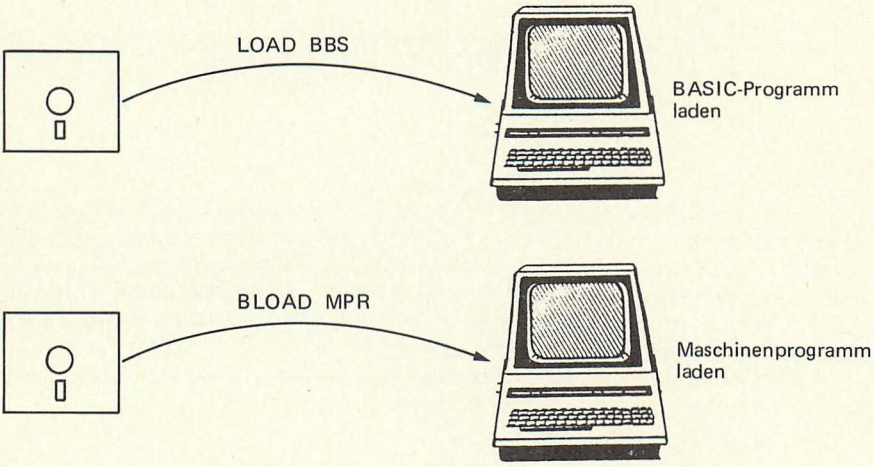
Zum Rückholen des Maschinenprogramms von der Diskette in den Speicher des APPLE-Computers genügt die Eingabe des folgenden Befehls:

BLOAD name

Nach Ausführung von BLOAD steht das Maschinenprogramm wieder an der gleichen Stelle im Speicher, an der wir es zuvor abgelegt hatten.

Der Befehl BLOAD kann über das Tastenfeld eingegeben werden oder in einem BASIC-Programm stehen.

Hier ein Anwendungsfall für die Tastenfeldeingabe von BLOAD:



Zuerst wird das Betriebssystem BBS, dann ein Maschinenprogramm geladen, um anschließend die Arbeit am Maschinenprogramm über das BBS fortzusetzen (siehe Abschnitt "Weitere Bearbeitung").

Und hier ein Anwendungsfall für BLOAD in einem BASIC-Programm:

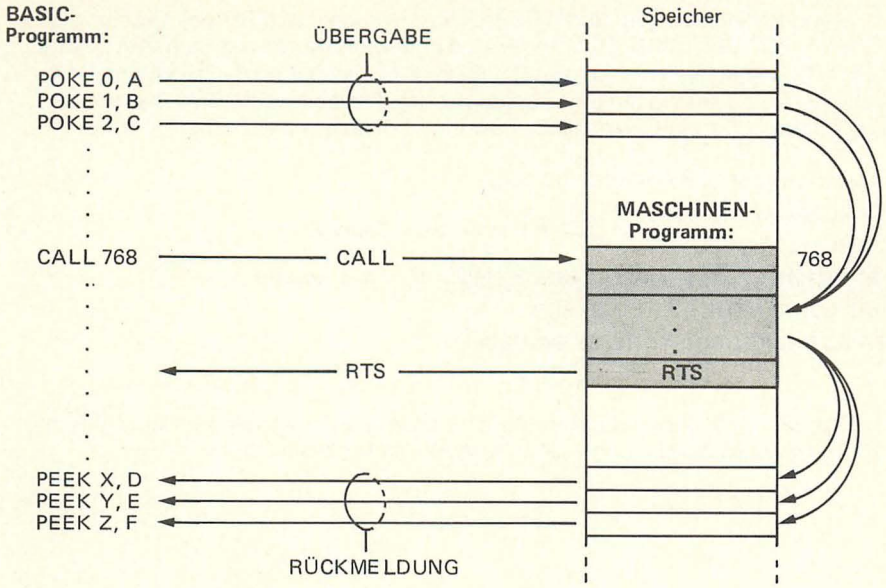
```
100 D$ = CHR$ (4): REM CTRL–D      <-- Laden des Maschinen-
110 PRINT D$; "BLOAD MPR"               programms MPR
 .
 .                                  } weitere Zeilen Ihres BASIC-Programms
 .
200 CALL 768                        <-- Aufrufen des Maschinenprogramms MPR
 .
 .                                  } weitere Zeilen Ihres BASIC-Programms
 .
```

In Programmform kann BLOAD nur in einem PRINT-Befehl angegeben werden. Hier lädt sich ein BASIC-Programm während des Ablaufs ein Maschinenprogramm, um es anschließend mit CALL 768 wie ein BASIC-Unterprogramm aufzurufen.

Aufrufen durch BASIC-Programme

Alle Maschinenprogramme, die wir in diesem Buch zu entwickeln gelernt haben, können in jedem BASIC-Haupt- (oder Unter-)programm als Unterprogramm dienen. (Laden und Aufrufen eines Maschinenprogramms zeigte der vorige Abschnitt).

Den Datenverkehr zwischen BASIC- und MASCHINEN-Programm kennen Sie inzwischen:



Weitere Bearbeitung

Wenn wir die Arbeiten am Maschinenprogramm unterbrechen, den APPLE-Computer ausschalten und später weiterarbeiten wollen, können wir das Betriebssystem BBS um folgende Programmzeilen erweitern:

```
110   PRINT "ANGABEN ZUM MASCHINEN
      PROGRAMM:"
115   PRINT
120   INPUT "ANFANGSADRESSE  = ";S

125   PRINT
130   INPUT "ANZAHL DER BYTES= ";B

131   PRINT
132   PRINT "NEUEINGABE ODER FORTS
      ETZUNG? N/F : "
133   PRINT : INPUT A$
134   IF A$ <  > "N" AND A$ <  > "
      F" THEN  GOTO 132
135   IF A$ = "N" THEN  GOTO 140
136   IF A$ = "F" THEN  GOTO 400
140   PRINT "MASCHINENPROGRAMM EIN
      GEBEN: "
```

Vor dem Ausschalten des APPLE-Computers speichern wir das Maschinenprogramm mit BSAVE auf der Diskette ab und holen es, ehe wir mit dem Betriebssystem BBS zu arbeiten beginnen, mit dem Befehl BLOAD wieder an die gleiche Stelle im Speicher zurück. Dann laden und starten wir das Betriebssystem BBS, das uns im Dialog fragt, ob wir eine Neueingabe oder eine Fortsetzung wünschen.

```
ANGABEN ZUM MASCHINENPROGRAMM:
ANFANGSADRESSE = 768          <= Ihre Angaben (Beispiel)
ANZAHL DER BYTES = 27
NEUEINGABE ODER FORTSETZUNG ? N/F: F <- Ihre Antwort
KEINE KORREKTUR:  99 EINGEBEN
FUER KORREKTUR:   ADRESSE EINGEBEN
```

Das BBS stellt Ihnen dann Ihr Maschinenprogramm wieder auf dem Bildschirm dar und gibt Ihnen Möglichkiet, seine Bearbeitung fortzusetzen.

ANHANG A
Programme im APPLE

A-1 Fest eingespeicherte Maschinenprogramme

Anfangsadresse im Speicher dez	hex	Funktion	im Buch Seite
63 488	F800	PUNKT DARSTELLEN	
63 513	F819	HORIZ. LINIE DARSTELLEN	
63 528	F828	VERT. LINIE DARSTELLEN	
64 320	FB40	GRAPHIK MODE SETZEN	
64 600	FC58	BILDSCHIRM LÖSCHEN	
64 795	FD1B	ZUFALLSZAHL ZIEHEN	
64 821	FD35	TASTENEINGABE LESEN	
64 910	FD8E	SPRUNG AUF FOLGENDEN ZEILENANFANG	
64 986	FDDA	AKKUMULATORINHALT HEX. DARSTELLEN	
65 005	FDED	AKKUMULATORINHALT ASCII DARSTELLEN	
65 338	FF3A	SIGNALTON DES LAUTSPRECHERS	

A-2 Bildschirmsymbole der Betriebsprogramme

Symbol	Bedeutung
*	SYSTEM MONITOR
]	APPLESOFT II
>	INTEGER BASIC
!	MINI-ASSEMBLER
■	CURSOR

ANHANG B
Tabellen für Verzweigungsbefehle

Vorverzweigungen

Verzweigungsschritte dez	hex
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	08
9	09
10	0A
11	0B
12	0C
13	0D
14	0E
15	0F
16	10
17	11
18	12
19	13
20	14
21	15
22	16
23	17
24	18
25	19
26	1A
27	1B
28	1C
29	1D
30	1E
31	1F
32	20
33	21
34	22
35	23
36	24
37	25
38	26
39	27
40	28
41	29
42	2A
43	2B
44	2C
45	2D
46	2E
47	2F
48	30

Verzweigungsschritte dez	hex
49	31
50	32
51	33
52	34
53	35
54	36
55	37
56	38
57	39
58	3A
59	3B
60	3C
61	3D
62	3E
63	3F
64	40
65	41
66	42
67	43
68	44
69	45
70	46
71	47
72	48
73	49
74	4A
75	4B
76	4C
77	4D
78	4E
79	4F
80	50
81	51
82	52
83	53
84	54
85	55
86	56
87	57
88	58
89	59
90	5A
91	5B
92	5C
93	5D
94	5E
95	5F
96	60

Verzweigungsschritte dez	hex
97	61
98	62
99	63
100	64
101	65
102	66
103	67
104	68
105	69
106	6A
107	6B
108	6C
109	6D
110	6E
111	6F
112	70
113	71
114	72
115	73
116	74
117	75
118	76
119	77
120	78
121	79
122	7A
123	7B
124	7C
125	7D
126	7E
127	7F

ANHANG B (Forts.)

Rückverzweigungen

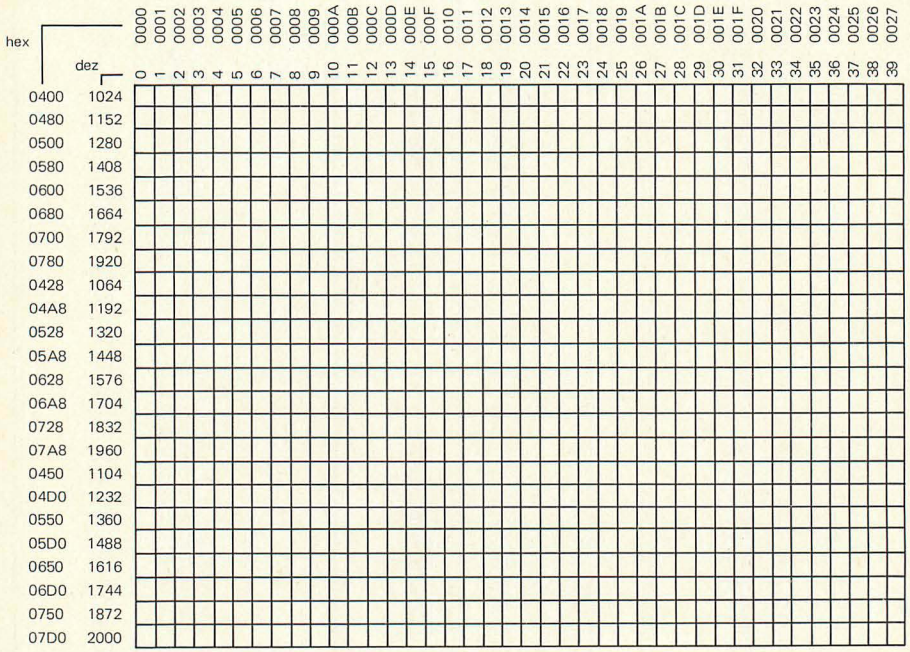
Verzweigungsschritte dez	hex
–1	FF
–2	FE
–3	FD
–4	FC
–5	FB
–6	FA
–7	F9
–8	F8
–9	F7
–10	F6
–11	F5
–12	F4
–13	F3
–14	F2
–15	F1
–16	F0
–17	EF
–18	EE
–19	ED
–20	EC
–21	EB
–22	EA
–23	E9
–24	E8
–25	E7
–26	E6
–27	E5
–28	E4
–29	E3
–30	E2
–31	E1
–32	E0
–33	DF
–34	DE
–35	DD
–36	DC
–37	DB
–38	DA
–39	D9
–40	D8
–41	D7
–42	D6
–43	D5
–44	D4
–45	D3
–46	D2
–47	D1
–48	D0

Verzweigungsschritte dez	hex
–49	CF
–50	CE
–51	CD
–52	CC
–53	CB
–54	CA
–55	C9
–56	C8
–57	C7
–58	C6
–59	C5
–60	C4
–61	C3
–62	C2
–63	C1
–64	C0
–65	BF
–66	BE
–67	BD
–68	BC
–69	BB
–70	BA
–71	B9
–72	B8
–73	B7
–74	B6
–75	B5
–76	B4
–77	B3
–78	B2
–79	B1
–80	B0
–81	AF
–82	AE
–83	AD
–84	AC
–85	AB
–86	AA
–87	A9
–88	A8
–89	A7
–90	A6
–91	A5
–92	A4
–93	A3
–94	A2
–95	A1
–96	A0

Verzweigungsschritte dez	hex
–97	9F
–98	9E
–99	9D
–100	9C
–101	9B
–102	9A
–103	99
–104	98
–105	97
–106	96
–107	95
–108	94
–109	93
–110	92
–111	91
–112	90
–113	8F
–114	8E
–115	8D
–116	8C
–117	8B
–118	8A
–119	89
–120	88
–121	87
–122	86
–123	85
–124	84
–125	83
–126	82
–127	81
–128	80

ANHANG C
Informationen für Bildschirmdarstellungen

C-1 Adressen zum 40 x 24-Bildschirmraster



Beispiel:
Speicheradresse von Zeichenfeld 17 in der ersten Zeile:
1024 + 16 = 1040 dez oder 0400 + 0010 = 0410 hex

C-2 Hex-Adressen des 40spaltigen Bildschirms

Zeilen	Spalten 1 . . . 40	
1	0400 →	0427
2	0480 →	04A7
3	0500 →	0527
4	0580 →	05A7
5	0600 →	0627
6	0680 →	06A7
7	0700 →	0727
8	0780 →	07A7
9	0428 →	044F
10	04A8 →	04CF
11	0528 →	054F
12	05A8 →	05CF
13	0628 →	064F
14	06A8 →	06CF
15	0728 →	074F
16	07A8 →	07CF
17	0450 →	0477
18	04D0 →	04F7
19	0550 →	0577
20	05D0 →	05F7
21	0650 →	0677
22	06D0 →	06F7
23	0750 →	0777
24	07D0 →	07F7

C-3 ASCII-Codes der Bildschirmzeichen

Zeichen	Darstellung		
	nor	bl	inv
	A0	60	20
!	A1	61	21
"	A2	62	22
#	A3	63	23
$	A4	64	24
%	A5	65	25
&	A6	66	26
'	A7	67	27
(	A8	68	28
)	A9	69	29
*	AA	6A	2A
+	AB	6B	2B
,	AC	6C	2C
–	AD	6D	2D
.	AE	6E	2E
/	AF	6F	2F
0	B0	70	30
1	B1	71	31
2	B2	72	32
3	B3	73	33
4	B4	74	34
5	B5	75	35
6	B6	76	36
7	B7	77	37
8	B8	78	38
9	B9	79	39
:	BA	7A	3A
;	BB	7B	3B
<	BC	7C	3C
=	BD	7D	3D
>	BE	7E	3E
?	BF	7F	3F

Zeichen	Darstellung		
	nor	bl	inv
@	C0	40	00
A	C1	41	01
B	C2	42	02
C	C3	43	03
D	C4	44	04
E	C5	45	05
F	C6	46	06
G	C7	47	07
H	C8	48	08
I	C9	49	09
J	CA	4A	0A
K	CB	4B	0B
L	CC	4C	0C
M	CD	4D	0D
N	CE	4E	0E
O	CF	4F	0F
P	D0	50	10
Q	D1	51	11
R	D2	52	12
S	D3	53	13
T	D4	54	14
U	D5	55	15
V	D6	56	16
W	D7	57	17
X	D8	58	18
Y	D9	59	19
Z	DA	5A	1A
[	DB	5B	1B
\	DC	5C	1C
]	DD	5D	1D
^	DE	5E	1E
–	DF	5F	1F

nor normal
bl blinkend
inv invertiert

Alle Code-Angaben hexadezimal

C4 Farbcodes

Farbcode dez	Farbcode hex	Farbe
0	0	Schwarz
1	1	Magenta
2	2	Dunkelblau
3	3	Hellpurpur
4	4	Dunkelgrün
5	5	Grau
6	6	Mittelblau
7	7	Hellblau
8	8	Braun
9	9	Orange
10	A	Grau
11	B	Pink
12	C	Grün
13	D	Gelb
14	E	Blau/Grün
15	F	Weiß

Farbcodes für den niedrigauflösenden Graphikmode.

ANHANG D
Maschinenbefehle des 6502

Befehl	Adressierungsart	Assemblerform	Code hex	Code dez	Bytes	Statusregister N	V	D	I	Z	C	
ADC	Unmittelbar	ADC #oper	69	105	2	x	x			x	x	
	Seite Null	ADC oper	65	101	2							
	Seite Null,X	ADC oper,X	75	117	2							
	Absolut	ADC oper	6D	109	3							
	Absolut, X	ADC oper,X	7D	125	3							
	Absolut,Y	ADC oper,Y	79	121	3							
	(Indirekt,X)	ADC (oper,X)	61	97	2							
	(Indirekt),Y	ADC (oper),Y	71	113	2							
AND	Unmittelbar	AND #oper	29	41	2	x				x		
	Seite Null	AND oper	25	37	2							
	Seite Null,X	AND oper,X	35	53	2							
	Absolut	AND oper	2D	45	3							
	Absolut,X	AND oper,X	3D	61	3							
	Absolut,Y	AND oper,Y	39	57	3							
	(Indirekt,X)	AND (oper,X)	21	33	2							
	(Indirekt),Y	AND (oper),Y	31	49	2							
ASL	Akkumulator	ASL A	0A	10	1	x				x	x	
	Seite Null	ASL oper	06	6	2							
	Seite Null,X	ASL oper,X	16	22	2							
	Absolut	ASL oper	0E	14	3							
	Absolut,X	ASL oper,X	1E	30	3							

Befehl	Adressierungsart	Assemblerform	Code hex	Code dez	Bytes	Statusregister N	V	D	I	Z	C	
BCC	Relativ	BCC oper	90	144	2							
BCS	Relativ	BCS oper	B0	176	2							
BEQ	Relativ	BEQ oper	F0	240	2							
BIT	Seite Null	BIT oper	24	36	2	x	x			x		
	Absolut	BIT oper	2C	44	3							
BMI	Relativ	BMI oper	30	48	2							
BNE	Relativ	BNE oper	D0	208	2							
BPL	Relativ	BPL oper	10	16	2							
BRK	Implizit	BRK	00	0	1				x			
BVC	Relativ	BVC oper	50	80	2							
BVS	Relativ	BVS oper	70	112	2							
CLC	Implizit	CLC	18	24	1						x	
CLD	Implizit	CLD	D8	216	1			x				
CLI	Implizit	CLI	58	88	1				x			
CLV	Implizit	CLV	B8	184	1		x					
CMP	Unmittelbar	CMP #oper	C9	201	2	x				x	x	
	Seite Null	CMP oper	C5	197	2							
	Seite Null,X	CMP oper,X	D5	213	2							
	Absolut	CMP oper	CD	205	3							
	Absolut,X	CMP oper,X	DD	221	3							
	Absolut,Y	CMP oper,Y	D9	217	3							

ANHANG D (Forts.)

Befehl	Adressierungsart	Assemblerform	Code hex	Code dez	Bytes	Statusregister N	V	D	I	Z	C	
	(Indirekt,X)	CMP (oper,X)	C1	193	2							
	(Indirekt),Y!	CMP (oper),Y	D1	209	2							
CPX	Unmittelbar	CPX #oper	E0	224	2	x				x	x	
	Seite Null	CPX oper	E4	228	2							
	Absolut	CPX oper	EC	236	3							
CPY	Unmittelbar	CPY #oper	C0	192	2	x				x	x	
	Seite Null	CPY oper	C4	196	2							
	Absolut	CPY oper	CC	204	3							
DEC	Seite Null	DEC oper	C6	198	2	x				x		
	Seite Null,X	DEC oper,X	D6	214	2							
	Absolut	DEC oper	CE	206	3							
	Absolut,X	DEC oper,X	DE	222	3							
DEX	Implizit	DEX	CA	202	1	x				x		
DEY	Implizit	DEY	88	136	1	x				x		
EOR	Unmittelbar	EOR #oper	49	73	2	x				x		
	Seite Null	EOR oper	45	69	2							
	Seite Null,X	EOR oper,X	55	85	2							
	Absolut	EOR oper	4D	77	3							
	Absolut,X	EOR oper,X	5D	93	3							
	Absolut,Y	EOR oper,Y	59	89	3							
	(Indirekt,X)	EOR (oper,X)	41	65	2							
	(Indirekt),Y	EOR (oper),Y	51	81	2							

Befehl	Adressierungsart	Assemblerform	Code hex	Code dez	Bytes	Statusregister N	V	D	I	Z	C	
INC	Seite Null	INC oper	E6	230	2	x				x		
	Seite Null,X	INC oper,X	F6	246	2							
	Absolut	INC oper	EE	238	3							
	Absolut,X	INC oper,X	FE	254	3							
INX	Implizit	INX	E8	232	1	x				x		
INY	Implizit	INY	C8	200	1	x				x		
JMP	Absolut	JMP oper	4C	76	3							
	Indirekt	JMP (oper)	6C	108	3							
JSR	Absolut	JSR oper	20	32	3							
LDA	Unmittelbar	LDA #oper	A9	169	2	x				x		
	Seite Null	LDA oper	A5	165	2							
	Seite Null,X	LDA oper,X	B5	181	2							
	Absolut	LDA oper	AD	173	3							
	Absolut,X	LDA oper,X	BD	189	3							
	Absolut,Y	LDA oper,Y	B9	185	3							
	(Indirekt,X)	LDA (oper,X)	A1	161	2							
	(Indirekt),Y	LDA (oper),Y	B1	177	2							
LDX	Unmittelbar	LDX #oper	A2	162	2	x				x		
	Seite Null	LDX oper	A6	166	2							
	Seite Null,Y	LDX oper,Y	B6	182	2							
	Absolut	LDX oper	AE	174	3							
	Absolut,Y	LDX oper,Y	BE	190	3							

ANHANG D (Forts.)

Befehl	Adressierungsart	Assemblerform	Code hex	Code dez	Bytes	Statusregister N	V	D	I	Z	C
LDY	Unmittelbar	LDY #oper	A0	160	2	x				x	
	Seite Null	LDY oper	A4	164	2						
	Seite Null,X	LDY oper,X	B4	180	2						
	Absolut	LDY oper	AC	172	3						
	Absolut,X	LDY oper,X	BC	188	3						
LSR	Akkumulator	LSR A	4A	74	1	x				x	x
	Seite Null	LSR oper	46	70	2						
	Seite Null,X	LSR oper,X	56	86	2						
	Absolut	LSR oper	4E	78	3						
	Absolut,X	LSR oper,X	5E	94	3						
NOP	Implizit	NOP	EA	234	1						
ORA	Unmittelbar	ORA #oper	09	9	2	x				x	
	Seite Null	ORA oper	05	5	2						
	Seite Null,X	ORA oper,X	15	21	2						
	Absolut	ORA oper	0D	13	3						
	Absolut,X	ORA oper,X	1D	29	3						
	Absolut,Y	ORA oper,Y	19	25	3						
	(Indirekt,X)	ORA (oper,X)	01	1	2						
	(Indirekt),Y	ORA (oper),Y	11	17	2						
PHA	Implizit	PHA	48	72	1						
PHP	Implizit	PHP	08	8	1						
PLA	Implizit	PLA	68	104	1	x				x	
PLP	Implizit	PLP	28	40	1						

Befehl	Adressierungsart	Assemblerform	Code hex	Code dez	Bytes	Statusregister N	V	D	I	Z	C	
ROL	Akkumulator	ROL A	2A	42	1	x				x	x	
	Seite Null	ROL oper	26	38	2							
	Seite Null,X	ROL oper,X	36	54	2							
	Absolut	ROL oper	2E	46	3							
	Absolut,X	ROL oper,X	3E	62	3							
ROR	Akkumulator	ROR A	6A	106	1	x				x	x	
	Seite Null	ROR oper	66	102	2							
	Seite Null,X	ROR oper,X	76	118	2							
	Absolut		6E	110	3							
	Absolut,X		7E	126	3							
RTI	Implizit	RTI	40	64	1							
RTS	Implizit	RTS	60	96	1							
SBC	Unmittelbar	SBC #oper	E9	233	2	x	x			x	x	
	Seite Null	SBC oper	E5	229	2							
	Seite Null,X	SBC oper,X	F5	245	2							
	Absolut	SBC oper	ED	237	3							
	Absolut,X	SBC oper,X	FD	253	3							
	Absolut,Y	SBC (oper,Y)	F9	249	3							
	(Indirekt,X)	SBC (oper,X)	E1	225	2							
	(Indirekt),Y	SBC (oper),Y	F1	241	2							
SEC	Implizit	SEC	38	56	1						x	
SED	Implizit	SED	F8	248	1			x				
SEI	Implizit	SEI	78	120	1				x			

ANHANG D (Forts.)

Befehl	Adressierungsart	Assemblerform	Code hex	Code dez	Bytes	Statusregister N	V	D	I	Z	C	
STA	Seite Null	STA oper	85	133	2							
	Seite Null,X	STA oper,X	95	149	2							
	Absolut	STA oper	8D	141	3							
	Absolut,X	STA oper,X	9D	157	3							
	Absolut,Y	STA oper,Y	99	153	3							
	(Indirekt,X)	STA (oper,X)	81	129	2							
	(Indirekt),Y	STA (oper),Y	91	145	2							
STX	Seite Null	STX oper	86	134	2							
	Seite Null,Y	STX oper,Y	96	150	2							
	Absolut	STX oper	8E	142	3							
STY	Seite Null	STY oper	84	132	2							
	Seite Null,X	STY oper,X	94	148	2							
	Absolut	STY oper	8C	140	3							
TAX	Implizit	TAX	AA	170	1	x				x		
TAY	Implizit	TAY	A8	168	1	x				x		
TYA	Implizit	TYA	98	152	1	x				x		
TSX	Implizit	TSX	BA	186	1	x				x		
TXA	Implizit	TXA	8A	138	1	x				x		
TXS	Implizit	TXS	9A	154	1							

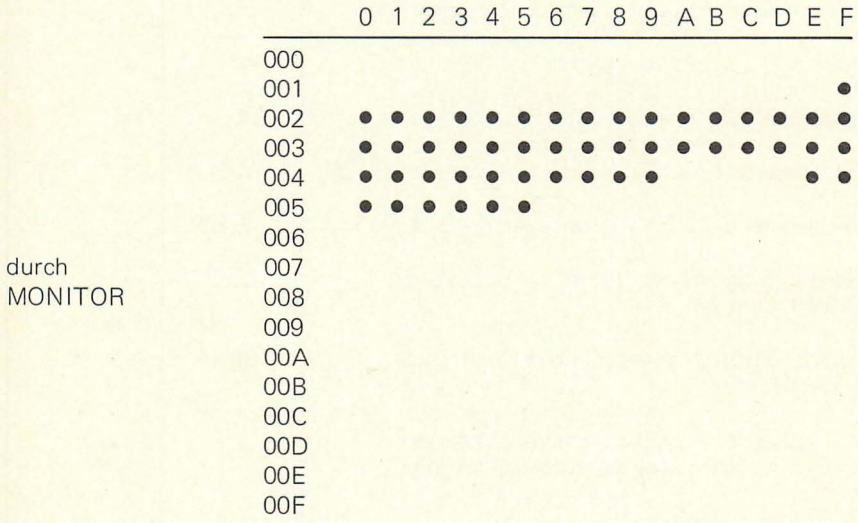
ANHANG E
Wichtige Informationen

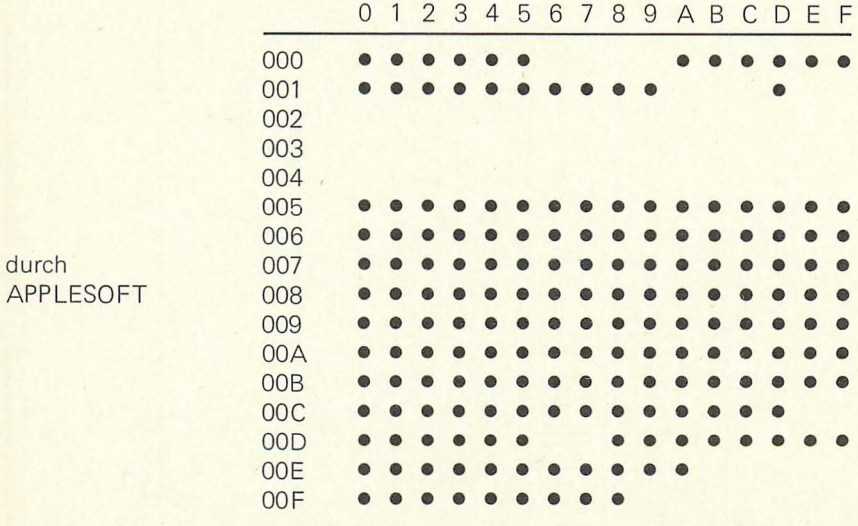
Beschreibung		im Buch Seite
Code für "Sprung auf Anfang der folgenden Zeile"	8D	
Code für "Leerzeichen (space)"	A0	
Speicheradresse für "Farbe eines Punkts"	0030	
Speicheradresse für "Ende einer horizontalen Linie"	002C	
Speicheradresse für "Ende einer vertikalen Linie"	002D	
Additionsprogramme: Statusbit C zu Anfang auf Wert 0 setzen	CLC	
Subtraktionsprogramme: Statusbit zu Anfang auf Wert 1 setzen	SEC	
Stapelspeicher: das Maschinenprogramm darf keinen Eintrag im Stapelspeicher zurücklassen		

ANHANG F

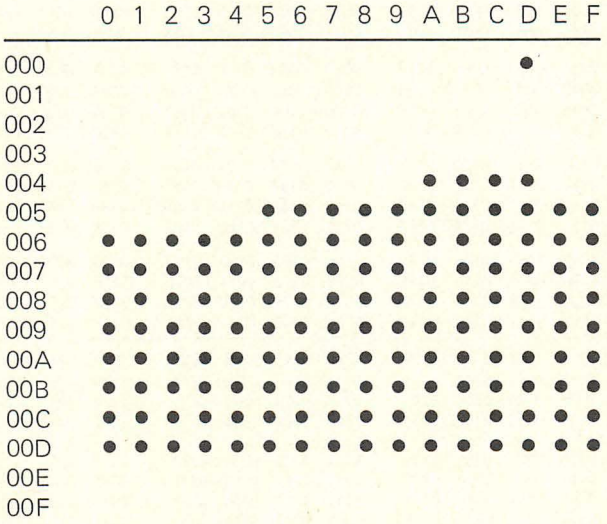
Belegung der Nullseite
Adressen 0000 . . . 00FF HEX

(Dargestellt für den APPLE IIe. Im Benutzerhandbuch jedes APPLE-Computers finden Sie die für Ihren Computertyp gültige Belegung.)

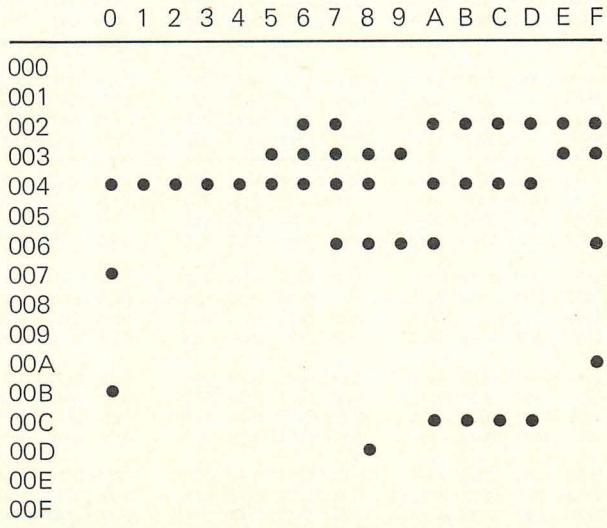




durch
INTEGER BASIC



durch
DOS 3.3



ANHANG G

Umwandlungstabellen dez–hex der Speicheradressen 0 . . . 65535

DEZ
0 . . . 767

HEX
0000 . . . 02FF

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
001	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
002	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
003	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
004	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
005	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
006	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
007	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
008	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
009	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
00A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
00B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
00C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
00D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
00E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
00F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
010	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
011	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
012	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
013	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
014	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
015	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
016	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
017	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
018	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
019	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
01A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
01B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
01C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
01D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
01E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
01F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
020	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
021	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
022	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
023	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
024	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
025	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
026	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
027	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
028	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
029	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
02A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
02B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
02C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
02D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
02E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
02F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767

	DEZ 768 . . . 1535								HEX 0300 . . . 05FF							
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
030	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
031	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
032	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
033	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0631
034	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
035	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
036	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
037	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
038	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
039	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
03A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
03B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
03C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
03D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
03E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
03F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
040	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
041	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
042	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
043	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
044	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
045	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
046	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
047	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
048	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
049	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
04A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
04B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
04C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
04D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
04E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
04F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
050	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
051	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
052	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
053	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
054	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
055	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
056	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
057	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
058	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
059	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
05A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
05B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
05C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
05D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
05E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
05F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

	DEZ 1536 . . . 2303								HEX 0600 . . . 08FF							
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
060	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
061	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
062	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
063	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
064	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
065	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
066	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
067	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1562	1663
068	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
069	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
06A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
06B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
06C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
06D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
06E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
06F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
070	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
071	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
072	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
073	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
074	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
075	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
076	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
077	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
078	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
079	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
07A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
07B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
07C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
07D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
07E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
07F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
080	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
081	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
082	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
083	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
084	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
085	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
086	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
087	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
088	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
089	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
08A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
08B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
08C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
08D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
08E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
08F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303

ANHANG G (Forts.)

	DEZ 2304 . . . 3071								HEX 0900 . . . 0BFF							
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
090	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
091	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
092	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
093	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
094	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
095	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
096	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
097	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
098	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
099	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
09A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
09B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
09C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
09D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
09E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
09F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
0A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
0A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
0A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
0A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
0A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
0A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
0A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
0A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
0A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
0A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
0AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
0AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
0AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
0AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
0AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
0AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
0B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
0B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
0B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
0B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
0B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
0B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
0B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
0B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
0B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
0B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
0BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
0BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
0BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
0BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
0BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
0BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

ANHANG G (Forts.)

	DEZ 3072 . . . 3839								HEX 0C00 . . . 0EFF							
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
0C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
0C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
0C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
0C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
0C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
0C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
0C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
0C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
0C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	2231
0CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
0CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
0CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
0CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
0CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
0CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
0D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
0D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
0D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
0D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
0D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
0D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
0D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
0D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
0D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
0D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
0DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
0DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
0DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
0DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
0DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
0DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
0E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
0E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
0E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
0E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
0E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
0E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
0E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
0E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
0E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
0E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
0EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
0EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
0EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
0ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
0EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
0EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839

ANHANG G (Forts.)

	DEZ 3840 . . . 4095								HEX 0F00 . . . 0FFF							
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
0F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
0F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
0F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
0F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
0F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
0F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
0F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
0F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
0F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
0FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
0FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
0FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
0FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
0FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
0FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

DEZ 4096 . . . 65 535	HEX 1000 . . . FFFF
dez	aus ob. Tabellen
4 096 . . . 8 191	1000 + (0000 . . . 0FFF)
8 192 . . . 12 287	2000 + (0000 . . . 0FFF)
12 288 . . . 16 383	3000 + (0000 . . . 0FFF)
16 384 . . . 20 479	4000 + (0000 . . . 0FFF)
20 480 . . . 24 575	5000 + (0000 . . . 0FFF)
24 576 . . . 28 671	6000 + (0000 . . . 0FFF)
28 672 . . . 32 767	7000 + (0000 . . . 0FFF)
32 768 . . . 36 863	8000 + (0000 . . . 0FFF)
36 864 . . . 40 959	9000 + (0000 . . . 0FFF)
40 960 . . . 45 055	A000 + (0000 . . . 0FFF)
45 056 . . . 49 151	B000 + (0000 . . . 0FFF)
49 152 . . . 53 247	C000 + (0000 . . . 0FFF)
53 248 . . . 57 343	D000 + (0000 . . . 0FFF)
57 344 . . . 61 439	E000 + (0000 . . . 0FFF)
61 440 . . . 65 535	F000 + (0000 . . . 0FFF)

Beispiel: 37 000 dez → 9088 hex
37 000 – 36 864 = 136 dez = 0088 hex
9000 + 0088 = 9088 hex

ANHANG H

Programm zur Hex/Dez-Wandlung

BASIC-Programm:

```
10  REM * HEX/DEZ WANDLUNG *
11  REM ********************
12  HOME
14  PRINT "**** HEX/DEZ WANDLUNG
     ****"
16  VTAB 10
20  PRINT "HEX="; SPC( 10)
22  GET A$: PRINT A$; SPC( 1)
24  GET B$: PRINT B$; SPC( 1)
26  GET C$: PRINT C$; SPC( 1)
28  GET D$: PRINT D$; SPC( 1)
30 A =  ASC (A$):B =  ASC (B$)
32 C =  ASC (C$):D =  ASC (D$)
40  IF A > 57 THEN A = A - 55: GOTO
     44
42 A = A - 48
44  IF B > 57 THEN B = B - 55: GOTO
     48
46 B = B - 48
48  IF C > 57 THEN C = C - 55: GOTO
     52
50 C = C - 48
52  IF D > 57 THEN D = D - 55: GOTO
     60
54 D = D - 48
60 DEZ = 4096 * A + 256 * B + 16 *
     C + D
70  PRINT : PRINT
80  PRINT  SPC( 35): HTAB 1
84  PRINT "DEZ="; SPC( 10);DEZ
90  GOTO 16
100  END
```

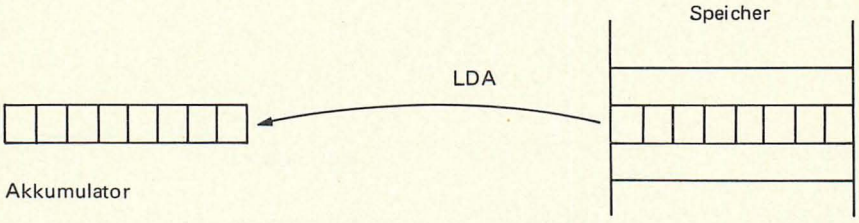
Bildschirmdarstellung

```
**** HEX/DEZ WANDLUNG ****

HEX=          F F F F     ← Ihre Eingabe

DEZ=          65535       ← Rückmeldung
```

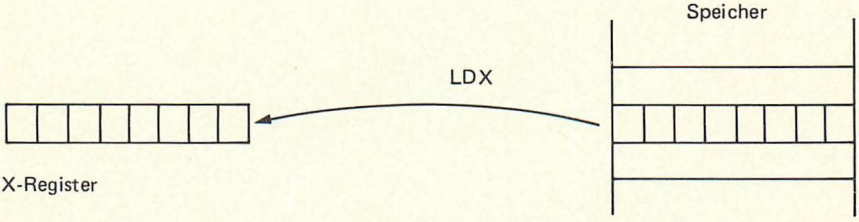
ANHANG I
Erläuterung der Befehle

Lade/Speicher- und Transfer-Befehle

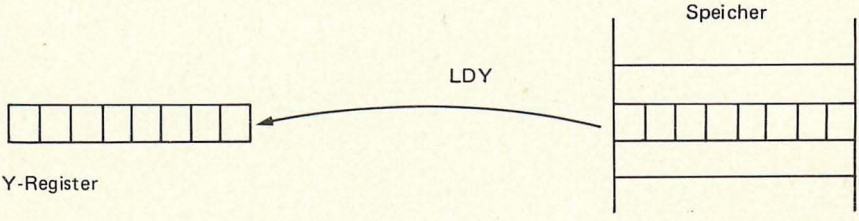
LDA – Lädt den Akkumulator vom Speicher (LoaD Accumulator). Der Inhalt einer Speicherstelle frei wählbarer Adresse wird in das Register "Akkumulator" auf dem 6502 gebracht.



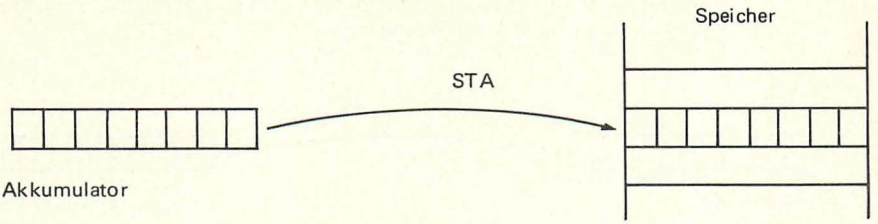
LDX – Lädt den Inhalt einer Speicherstelle wählbarer Adresse in das X-Register auf dem 6502. (LoaD X-register).



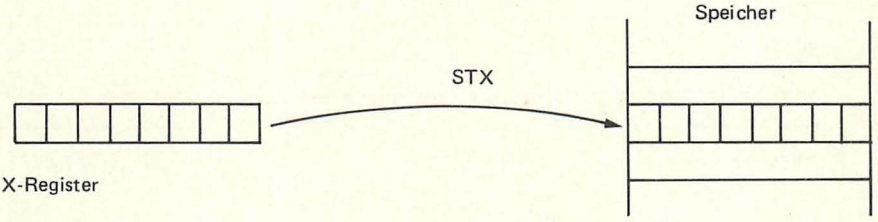
LDY – Lädt den Inhalt einer Speicherstelle wählbarer Adresse in das Y-Register auf dem 6502. (LoaD Y-register).



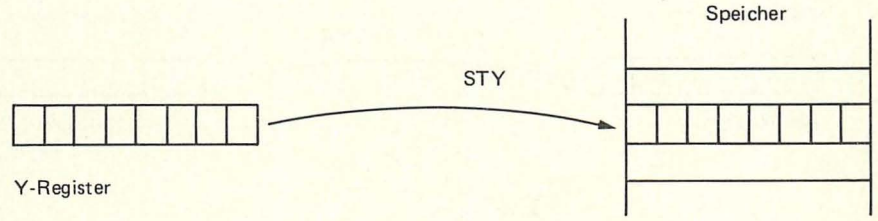
STA – Speichert den Inhalt des Akkumulators in eine Speicherstelle wählbarer Adresse. (STore Accumulator).



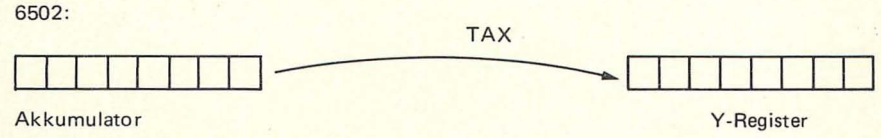
STX – Speichert den Inhalt des X-Registers in eine Speicherstelle wählbarer Adresse. (STore X-register).



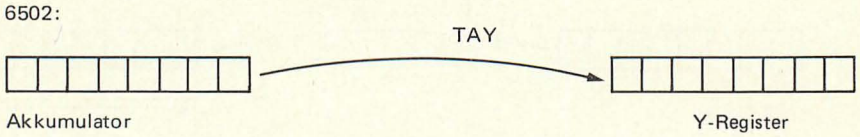
STY – Speichert den Inhalt des Y-Registers in eine Speicherstelle wählbarer Adresse. (STore Y-register).



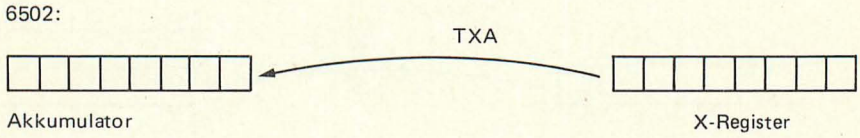
TAX – Transferiert den Inhalt des Akkumulators in das X-Register. (Transfer contents of Accumulator to X-register).



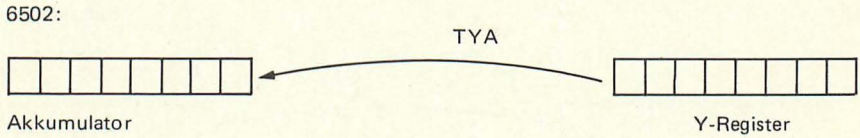
TAY – Transferiert den Inhalt des Akkumulators in das Y-Register. (Transfer contents of Accumulator to Y-register).



TXA – Transferiert den Inhalt des X-Registers in den Akkumulator. (Transfer contents of X-register to Accumulator).

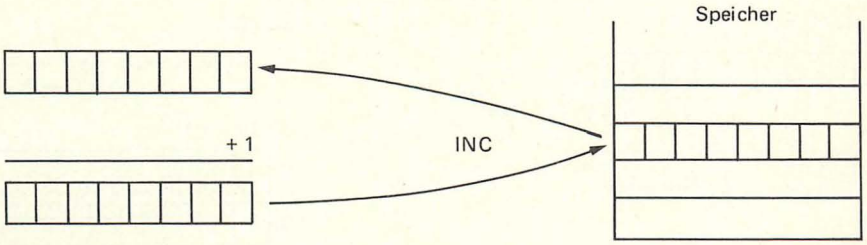


TYA – Transferiert den Inhalt des Y-Registers in den Akkumulator. (Transfer contents of Y-register to Accumulator).

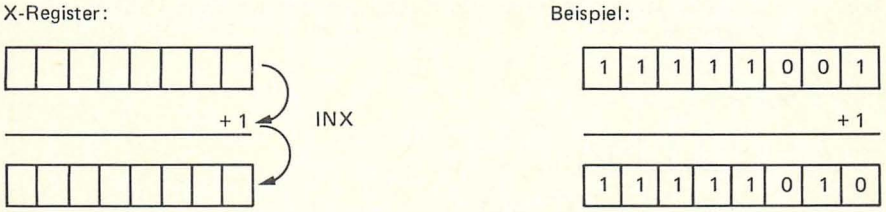


Befehle für Arithmetische Operationen

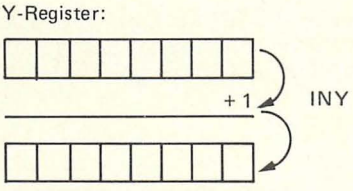
INC – Inkrementiert (erhöht) den Inhalt einer Speicherstelle frei wählbarer Adresse um den Wert 1. (INCrement).



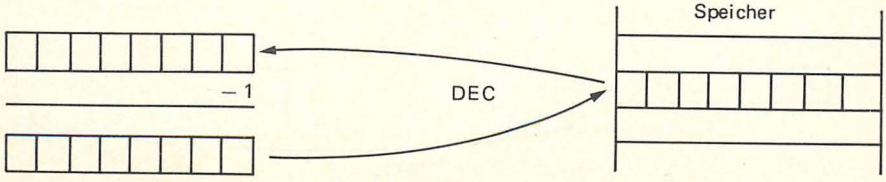
INX – Inkrementiert den Inhalt des X-Registers um den Wert 1.



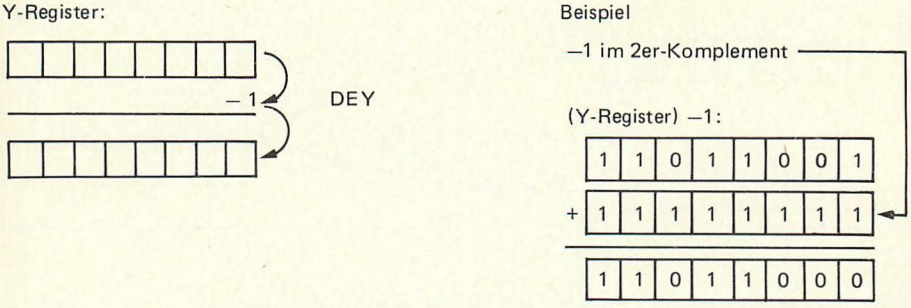
INY – Inkrementiert den Inhalt des Y-Registers um den Wert 1.



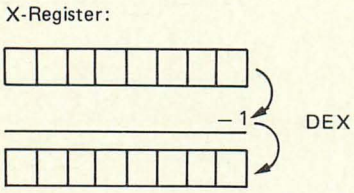
DEC – Dekrementiert (verringert) den Inhalt einer Speicherstelle wählbarer Adresse um den Wert 1. (DECrement).



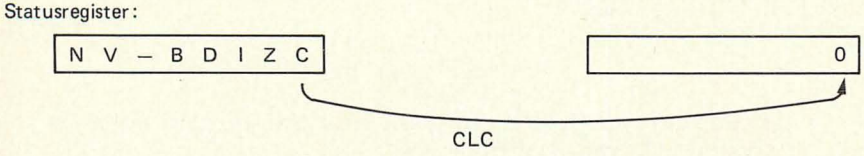
DEY – Dekrementiert den Inhalt des Y-Registers um den Wert 1.



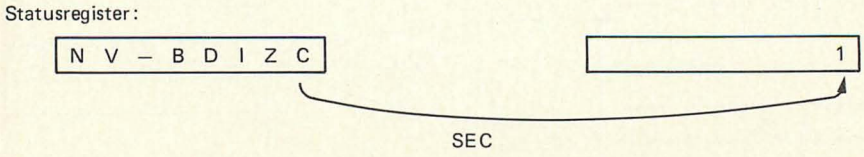
DEX – Dekrementiert den Inhalt des X-Registers um den Wert 1. (DECrement X-register).



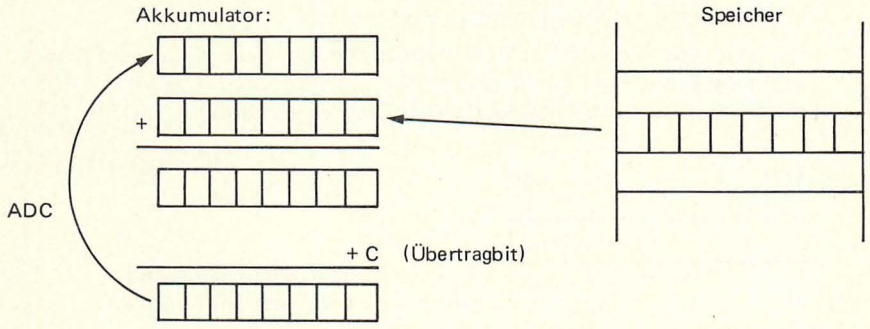
CLC – Löscht das Bit C (Übertragbit) im Statusregister (CLear Carry bit). Hierdurch werden Übertrag/Borgen in nachfolgender Addition/Subtraktion beeinflußt. Siehe Kapitel über ARITHMETIK und Beispiel in Kapitel 10. (CLear Carry bit).



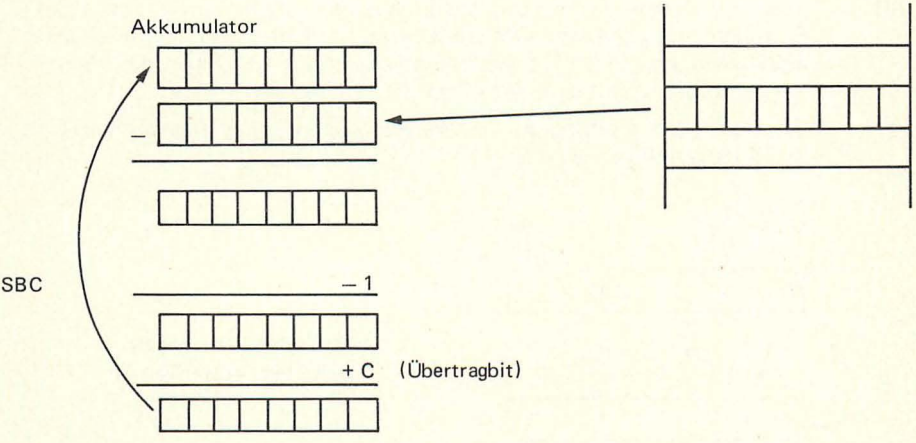
SEC – Setzt das Bit C (Übertragbit) auf den Wert 1. (SEt Carry bit). Gegenstück zu CLC.



ADC – Addiert den Inhalt des Akkumulators zum Inhalt einer Speicherstelle frei wählbarer Adresse; hierauf wird der momentane Wert von Bit C (Übertragbit) des Statusregisters addiert. (ADd immediate with Carry). Das Ergebnis wird im Akkumulator abgelegt.

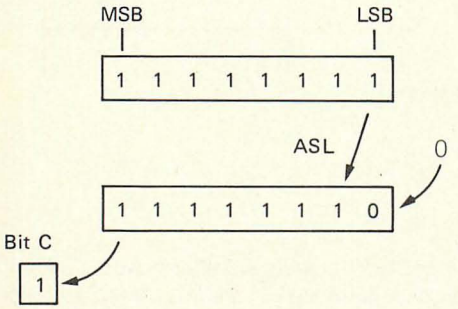


SBC – Subtrahiert zuerst den Inhalt einer Speicherstelle wählbarer Adresse vom Inhalt des Akkumulators und anschließend eine 1; addiert hierzu den Wert des Bit C (Übertragbit) im Statuswort und speichert das Ergebnis im Akkumulator. (Subtract immediate with Borrow, from Accumulator).



ASL – Arithmetische Linksverschiebung des Inhalts des Akkumulators oder einer Speicherstelle wählbarer Adresse. (Arithmetic Shift Left). Das höchstwertige Bit geht in Bit C (Übertragbit) des Statusregisters über; das niedrigstwertige Bit wird mit dem Wert 0 gefüllt.

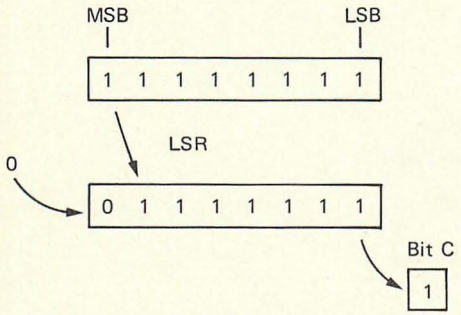
Die Linksverschiebung um 1 Stelle entspricht der Multiplikation des betreffenden Byte mit dem Wert 2; sie wird insbesondere in Algorithmen zur Multiplikation und Division eingesetzt (siehe Kapitel ARITHMETIK).



MSB höchstwertiges Bit
LSB niedrigstwertiges Bit

LSR – Logische Rechtsverschiebung des Inhalts des Akkumulators oder einer Speicherstelle wählbarer Adresse (Logical Shift Register). Das niedrigstwertige Bit geht in Bit C (Übertragbit) des Statusworts über; das höchstwertige Bit wird mit dem Wert 0 gefüllt.

Die Rechtsverschiebung um 1 Stelle entspricht der Division des betreffenden Byte durch den Wert 2.



MSB höchstwertiges Bit
LSB niedrigstwertiges Bit

Befehle für Verzweigungen, Schleifen und Unterprogrammsprünge

BCC – Verzweigen, wenn Bit C (Übertragbit) im Statusregister den Wert C = 0 hat. (Branch if Carry Clear). Das Programm verzweigt zu einer Speicheradresse, die **relativ** zur Speicheradresse des Befehls BCC verschoben liegt. Die Größe der Verschiebung wird im Operanden oper des Befehls BCCoper als positive oder negative Hexadezimalzahl angegeben. Siehe hierfür die Tabellen in Anhang B. Zu beachten: die Verschiebung der Speicheradresse um ±oper geht von der Speicheradresse des **nach** BCC folgenden Befehls aus. Da BCC 2 Speicheradressen belegt, berechnet sich die Verzweigungsadresse aus (1. Speicheradresse von BCC) +(2)±(oper).

BCS – Verzweigen, wenn Bit C im Statusregister den Wert C = 1 hat. (Branch if Carry Set). Die **relative** Verzweigung (siehe BCC) erfolgt nur, wenn C = 1 ist; andernfalls übergeht das Programm den Befehl BCS.

BEQ – Verzweigen, wenn Bit Z (Ergebnis Null) im Statusregister den Wert Z = 1 hat. (Branch if Equal). Bit Z wird auf den Wert 1 gesetzt, wenn eine vorangehende logische oder arithmetische Operation zum Ergebnis 00000000 im Akkumulator geführt hatte. Die Verzweigung erfolgt wie bei BCC **relativ** zur Speicheradresse von BEQ (siehe BCC).

BNE – Verzweigen, wenn Bit Z im Statusregister den Wert Z = 0 hat. (Branch on result Not Equal). Die **relative** Verzweigung (siehe BCC) erfolgt nur, wenn Z = 0 ist; andernfalls übergeht das Programm den Befehl BNE.

BPL – Verzweigen, wenn Bit N im Statusregister den Wert N = 0 hat. (Branch if result PLus). Bit N wird auf N = 0 gesetzt, wenn eine vorangehende logische oder arithmetische Operation zu einem positiven Ergebnis führte. (Für genaue Angabe der Operationen, die Bit N setzen oder rücksetzen, siehe Anhang D). Die Verzweigung erfolgt **relativ** zur Speicheradresse von BPL (siehe BCC). Hat Bit N den Wert N = 1, dann übergeht das Programm den Befehl BPL.

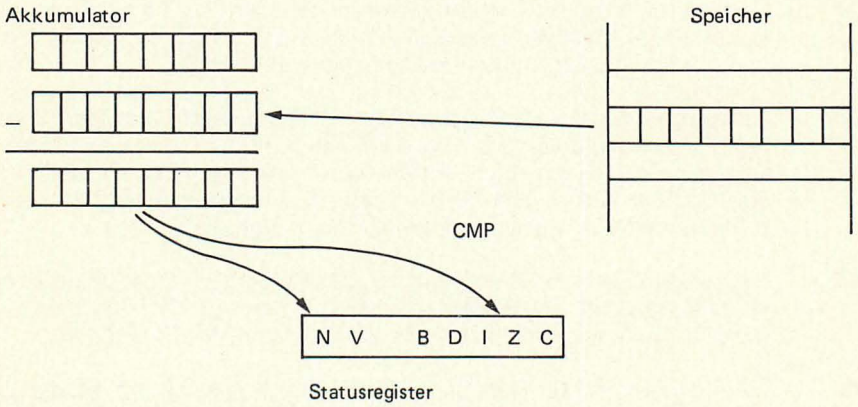
BMI – Verzweigen, wenn Bit N im Statusregister den Wert N = 1 hat. (Branch if result MInus). Komplement zu BPL.

JSR – Sprung in ein Unterprogramm, dessen erster Befehl unter der frei wählbaren, nach JSR stehenden Speicheradresse steht. (Jump to SubRoutine). Die Speicheradresse des auf JSR im aufrufenden Programm folgenden Befehls wird in den Stapelspeicher übertragen. Der letzte Befehl im Unterprogramm, RTS, setzt das aufrufende Programm mit der im Stapelspeicher stehenden Adresse fort. Das Befehlspaar JSR–RTS entspricht dem Befehlspaar GOSUB – RETURN in BASIC-Programmen.

RTS – Rückkehrbefehl aus einem Unterprogramm in das aufrufende Programm. (ReTurn from Subroutine). Die beiden zuletzt eingetragenen Bytes im Stapelspeicher werden zurückgelesen und in den Programmzähler übertragen. Siehe auch JSR.

CMP – Vergleiche den Inhalt einer Speicherstelle wählbarer Adresse mit dem Inhalt des Akkumulators und zeige ein Ergebnis NULL oder NEGATIV

durch Setzen der Bits Z (für Null) bzw. N (für negativ) im Statusregister an. (CoMPare). Dieser Befehl subtrahiert den Inhalt der adressierten Speicherstelle vom Inhalt des Akkumulators und zeigt das Ergebnis nur durch Setzen oder Nicht-setzen der Bits Z, N an; die Inhalte von Speicherstelle und Akkumulator bleiben unverändert.
Über einen nach CMP folgenden Vergleichsbefehl wie BEQ, BNE (fragen Bit Z ab) oder BPL, BMI (fragen Bit N ab) kann CMP indirekt eine Verzweigung auslösen.



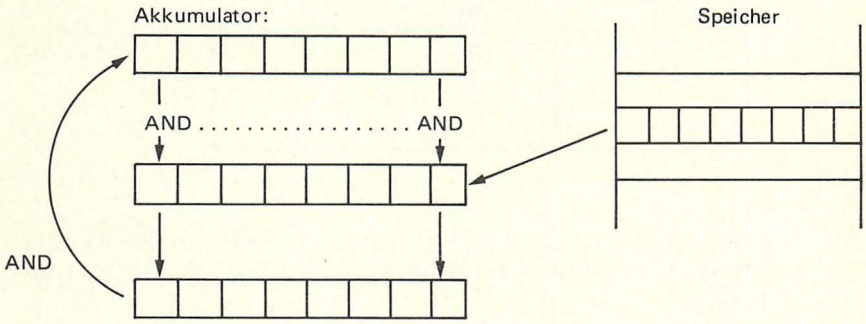
CPX – Vergleiche den Inhalt einer Speicherstelle wählbarer Adresse mit dem Inhalt des X-Registers und zeige ein Ergebnis NULL oder NEGATIV durch Setzen der Bits Z, N im Statusregister an. (ComPare immediate with X-register). Die Inhalte von Speicherstelle und Akkumulator bleiben unverändert. Die eigentliche Verzweigung wird mit nachfolgenden Verzweigungsbefehlen (BEQ, BNE und BPL, BMI) durch Abragen der Bits N und Z ausgelöst.

CPY – Wie CPX, jedoch mit dem Y-Register.

JMP – Programmsprung zu einer frei wählbaren, nach JMP angegebenen Speicheradresse. (JuMP). Der Inhalt dieser Speicherstelle muß einen ausführbaren Binärcode, also einen der Maschinenbefehle enthalten. Vergleichbar mit dem BASIC-Befehl GOTO.

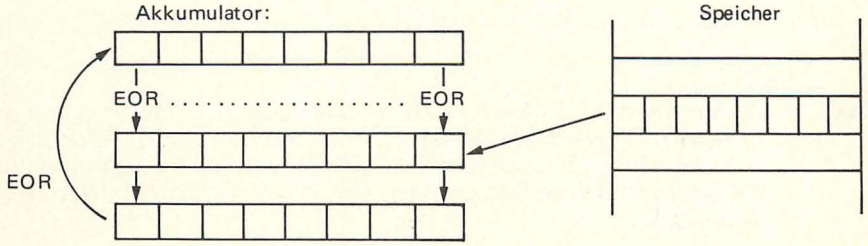
Befehle für Logische Operationen

AND – Logische UND-Verknüpfung des Inhalts einer Speicherstelle wählbarer Adresse und des Akkumulatorinhalts. Das Ergebnis wird im Akkumulator abgelegt.



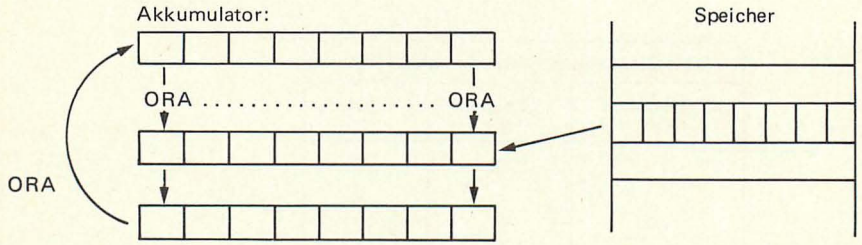
Wahrheitstabelle der AND-Verknüpfungen		
Akkumulator Bit	Speicher Bit	Ergebnis Bit
0	0	0
0	1	0
1	0	0
1	1	1

EOR – Logische EXCLUSIV-ODER-Verknüpfung des Inhalts einer Speicherstelle wählbarer Adresse und des Akkumulatorinhalts. Das Ergebnis steht im Akkumulator.



Wahrheitstabelle der EXCLUSIV-ODER-Verknüpfung		
Akkumulator Bit	Speicher Bit	Ergebnis Bit
0	0	0
0	1	1
1	0	1
1	1	0

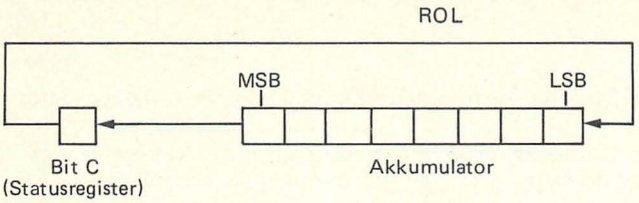
ORA – Logische ODER-Verknüpfung des Inhalts einer Speicherstelle wählbarer Adresse und des Inhalts des Akkumulators. Das Ergebnis steht im Akkumulator.



Wahrheitstabelle der ORA-Verknüpfung		
Akkumulator Bit	Speicher Bit	Ergebnis Bit
0	0	0
0	1	1
1	0	1
1	1	1

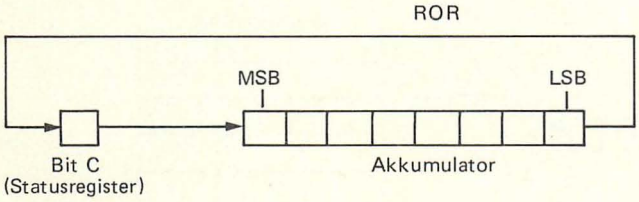
ASL – Arithmetische Linksverschiebung des Inhalts des Akkumulators oder einer Speicherstelle wählbarer Adresse. (Arithmetic Shift Left). Alle Bits im Byte werden um 1 Stelle nach links verschoben; das höchste Bit geht in das Übertragbit C im Statusregister über; in das niedrigste Bit wird der Wert 0 geladen.

ROL – Zyklische Linksverschiebung des Inhalts des Akkumulators oder einer Speicherstelle wählbarer Adresse über Bit C (Übertragbit) des Statusworts. (ROtate Left). Der momentane Inhalt von Bit C geht in das niedrigstwertige Bit des verschobenen Byte über; das höchstwertige Bit geht als neuer Inhalt nach Bit C über.



MSB höchstwertiges Bit
LSB niedrigstwertiges Bit

ROR – Zyklische Rechtsverschiebung des Inhalts von Akkumulator oder einer Speicherstelle wählbarer Adresse. (ROtate Right). Komplement zu ROL.

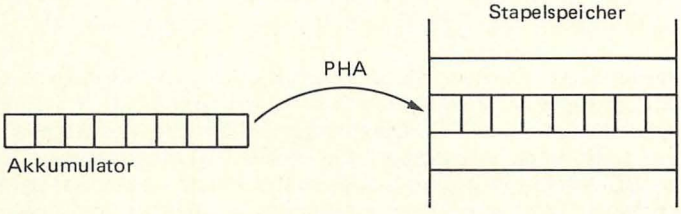


MSB höchstwertiges Bit
LSB niedrigstwertiges Bit

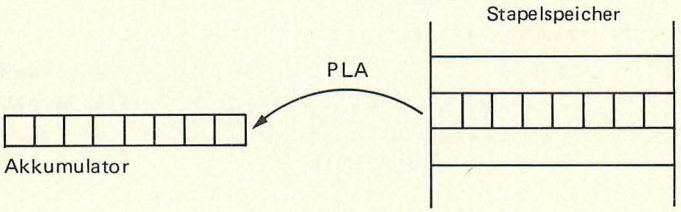
LSR – Logische Rechtsverschiebung des Inhalts des Akkumulators oder einer Speicherstelle frei wählbarer Adresse. (Logical Shift Right). Das niedrigstwertige Bit geht in Bit C (Übertragbit) des Statusregisters über; das höchstwertige Bit wird mit dem Wert 0 gefüllt.

Befehle zum Stapelspeicher

PHA – Übertrage den Akkumulatorinhalt in den Stapelspeicher. (PusH Accumulator onto stack). Das übertragene Byte wird auf den Stapel bereits vorhandener Bytes im Stapelspeicher geladen. Die Arbeitsweise des Stapelspeichers ist: Das zuletzt übertragene Byte wird von einem nachfolgenden Lesebefehl (siehe PLA) als erstes wieder in den Akkumulator zurückgebracht.



PLA – Hole 1 Byte vom Stapelspeicher in den Akkumulator. (PulL Accumulator off the stack).
Befehle zum Stapelspeicher dienen zur vorübergehenden Aufbewahrung von Akkumulatorinhalten – etwa in Programmabschnitten, in denen der Akkumulator abwechselnde Funktionen hat. Der Stapelspeicher wird automatisch vom Befehlspaar JSR – RTS benutzt (siehe JSR). Zwischen JSR und RTS liegende PHA, PLA-Befehle müssen paarweise auftreten und dürfen keine Bytes im Stapelspeicher liegen lassen, die vom RTS-Befehl fälschlich als Rücksprungadresse interpretiert werden.



Nicht behandelte Maschinenbefehle

BIT – UND-Verknüpfung des Inhalts des Akkumulators und einer Speicherstelle frei wählbarer Adresse; anders als bei AND wird das Ergebnis nur in Bit Z des Statusregisters angezeigt, falls das Bitmuster 00000000 resultierte. Die Inhalte von Akkumulator und Speicherstelle bleiben unverändert.

BVS – Verzweigen, wenn Bit O (Überlauf) im Statusregister den Wert O = 1 hat. (Branch on oVerflow Set). Die Verzweigung erfolgt **relativ** zur Speicheradresse des Befehls BVS (siehe BCC). Das Überlaufbit O wird von logischen und arithmetischen Operationen beeinflußt: tritt in Bit 6 einer dieser Operationen der Wert 1 auf, dann wird Bit O im Statusregister auf 1 gesetzt; hat Bit 6 den Wert 0, dann wird Bit O im Statusregister auf 0 zurückgesetzt.

BVC – Verzweigen, wenn Bit O (Überlauf) im Statusregister den Wert 0 = 1 hat. Komplement zu Befehl BVS. Hat Bit O den Wert O = Ø, dann übergeht das Programm diesen Befehl. (Branch on oVerflow Cleared).

CLD – Dezimalmode löschen. (CLear Decimal mode). Der Mikroprozessor 6502 kann arithmetische Operationen mit binär und mit BCD-dargestellten Daten ausführen. In BCD-Darstellung (BCD-Mode) werden alle Bytes als zwei 4-Bit-Worte interpretiert, die jeweils 1 Dezimalzahl 0 . . . 9 darstellen. BCD-Arithmetik wird in diesem Buch nicht behandelt; sie tritt besonders in Taschenrechnern auf und wird ausführlich in "Einführung in die Mikrocomputertechnik", A. Osborne, te-wi Verlag München, behandelt.

CLI – Interrupts unterdrücken. (CLear Interrupt requests). Der Mikroprozessor verfügt über einen Anschluß für externe Interrupt-Anforderungen (Pin IRQ). Mit dem Befehl CLI kann ein Programm verhindern, daß ein externes Gerät seinen Ablauf durch Anfordern eines Interrupts unterbricht.

CLV – Bit O im Statusregister löschen, d.h. auf den Wert 0 setzen. (CLear Overflow bit). Vergleichbar mit CLC.

PHP – Stapelspeichere den Inhalt des Statusregisters. (PusH Processor status). Der Inhalt des Statusregisters wird auf den Stapelspeicher geladen. Insbesondere bei Verzweigungen, die vom Inhalt des Statusregisters (oder einem seiner Bits) abhängen, kann es sinnvoll sein, die Statusbits vor ihrer Veränderung durch nachfolgende logische oder arithmetische Operationen zu retten. (Siehe folgenden Befehl zum Rückholen des Inhalts des Statusregisters).

PLP – Stapellade den Inhalt des Statusregisters. (PulL Processor status). Der Inhalt des Statusregisters wird nach vorangegangenem Abspeichern mit PHP wieder in das Statusregister zurückgeholt. Die augenblicklichen Werte der Bits, N, V, B, D, I, Z, C des Statusregisters werden durch die abgespeicherten und zurückgeholten Werte überschrieben.

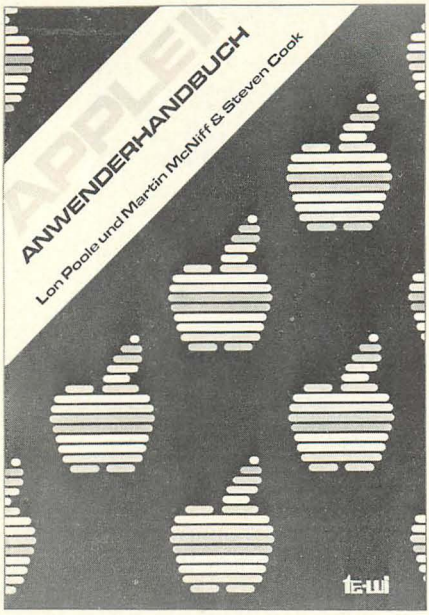
RTI – Rückkehr aus einer Interruptroutine. (ReTurn from Interrupt). Nach Annahme eines Interrupts springt der Mikroprozessor 6502 in ein Programm, das den Interrupt beantwortet. Am Ende dieser "Interruptroutine" bewirkt der Befehl RTI die Rückkehr in das unterbrochene Programm und dessen Fortsetzung.

SED – In Dezimalmode übergehen. Komplement zu CLD (siehe dort). (SEt Decimal mode).

TSX – Transferieren des Stapelspeicherzeigers in das X-Register. Der "Zeiger zum Stapelspeicher" enthält die Adresse der obersten, freien Speicherstelle im Stapelspeicher, d.h. einen der Adresswerte 0100 . . . 01FF in Seite Eins.

Register

APPLE II – ANWENDERHANDBUCH



Dieses deutschsprachige Apple II-Anwenderhandbuch erspart Ihnen zeitraubendes und nutzloses Suchen nach der wirklich verwendbaren Literatur über diesen beliebten Computer. Es beschreibt zum Einen den Apple II-Computer als solchen und gibt zum Anderen ausführlich Auskunft über Peripherie-Bausteine und Zubehör einschließlich Disk-Laufwerken und Drucker.
Mit Hilfe dieses Buches werden Sie Ihren Apple II erfolgreich einsetzen, denn der Informationsgehalt geht weit über das hinaus, was herstellerseitig an Dokumentation angeboten wird. Sie lernen BASIC auch für komplexe Anwendungen anzuwenden. Wie man farbige Grafiken erstellt. Sie erhalten Tips für fortgeschrittene Programmerstellung. Sie erfahren die Verwendung des Maschinensprachen-Monitors. U.v.m.
Mit dem Apple II-Anwenderhandbuch werden Ihnen alle Möglichkeiten eröffnet, die in diesem Computer stecken.
Die deutsche Übersetzung entstand in enger Zusammenarbeit mit der Herstellerfirma.

APPLE II – Anwenderhandbuch
von Lon Poole, McNiff & Steven Cook
400 Seiten, Paperback, DM 56.– Der angegebene Preis ist der Ladenpreis.

---

te-wi Verlag GmbH
Telefon 089/1292090

te-wi

Theo-Prosel-Weg 1
8000 München 40

VisiCalc – 50 populäre Anwendungen
– Die heutige Art, Daten transparent zu machen –

Dieses Buch befaßt sich mit dem Programmsystem VisiCalc, mit dem Sie Ihren Heimcomputer in einen "Elektronischen Berechnungsbogen" verwandeln können. Damit ist gemeint, daß elektronisch die Ausführung aller Berechnungen und die Darstellung der Ergebnisse auf dem Bildschirm erfolgt.
Und "Berechnungsbogen" steht für alle ausgebreiteten Zahlenwerke – wie etwa ein Formular zur Steuererklärung – in das nicht nur Zahlen einzutragen, sondern auch Summen, Überträge usw. zu bilden sind.
VisiCalc läßt die flüchtigen, fast verzögerungslos veränderbaren elektronischen Berechnungen und Darstellungen auf Heimcomputern für Planspiele mit Zahlen oder "Was-Wenn-Überlegungen" nutzen. Mit insgesamt 51 verschiedenen Modellen ist dieses Buch voll von derartigen Anwendungen in Wirtschaft und Privatleben. Ausführliche Erklärungen und Auflistungen machen es leicht, alle Programme nach persönlichen Bedürfnissen selbst abzuwandeln.
Der besondere Clou: eine dem Buch beigefügte 5 1/4-Zoll-Diskette, auf der alle Berechnungen und die Darstellungsformen gespeichert sind, erleichtern Ihnen das Arbeiten mit diesem Buch.
VisiCalc ist zum Inbegriff für die moderne Art geworden, Berechnungen auszuführen und anschaulich darzustellen.
Bei Bestellung Computertyp angeben.

VisiCalc – 50 populäre Anwendungen
von David M. Castlewitz und Lawrence J. Chisausky
184 Seiten, Paperback, Diskette
DM 79,–(der angegebene Preis ist der Ladenpreis)

---

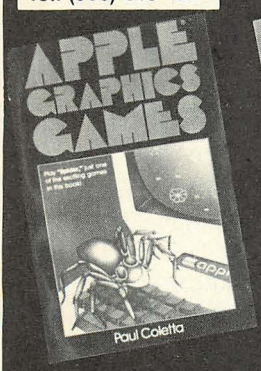
te-wi Verlag GmbH
Telefon 089/1292090

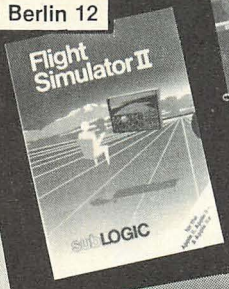
te-wi

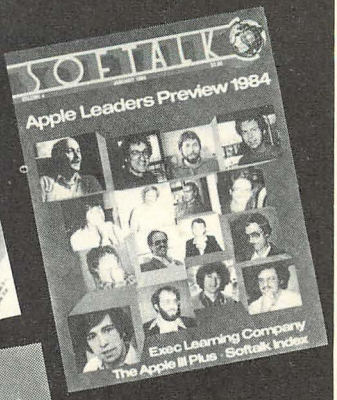
Theo-Prosel-Weg 1
8000 München 40

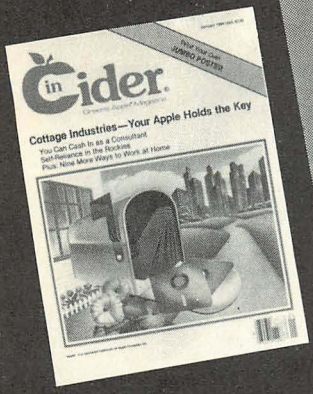
pandasoft

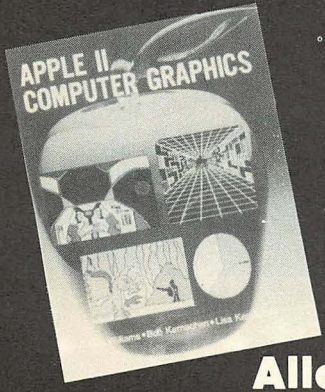
Uhlandstraße 195 (am Steinplatz) · D-1000 Berlin 12
Tel. (030) 310 423











Alles für den APPLE II
Hardware, Software, Bücher, Zeitschriften.
Fordern Sie unseren Gratis-Katalog an!

Bitte senden Sie mir Ihren kostenlosen Katalog.
Name: _____
Straße: _____
Ort: _____

6502 – PROGRAMMIEREN IN ASSEMBLER (Lance A.Leventhal)

Dieser Band, der eine ganze Serie von deutschsprachigen Büchern über die Programmierung in Assembler repräsentiert, enthält über 80 praktische Programmierbeispiele im Standardformat einschließlich Flußdiagrammen, Quellprogrammen, Objektcodes und erläuternden Texten für den Mikroprozessor 6502.

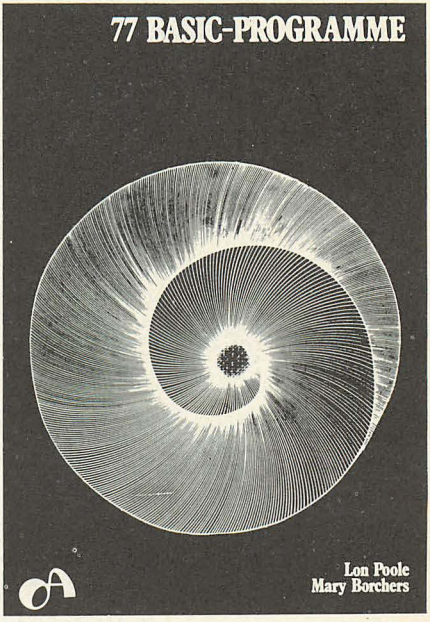
Ausführliche Besprechungen für die Erstellung von Programmen, von der Definition der Aufgabe ,über Testen, Fehlersuche, Dokumentation, bis hin zu modularer und strukturierter Programmierung runden dieses aktuelle Werk ab.

6502 – Programmieren in Assembler, 700 Seiten, Paperback, DM 59,–
Der angegebene Preis ist der Ladenpreis.

77 BASIC-Programme (Lon Poole und Mary Borchers)

Dieses Buch enthält eine Sammlung von praktischen 77 Kurzprogrammen, die in der populären Programmiersprache BASIC geschrieben sind. Es werden mathematische, finanztechnische, statistische und verschiedene allgemeine Aufgaben behandelt. Die leichtverständlich erläuterten Beispiele können direkt und selbst ohne jede Erfahrung in BASIC verwendet werden. Jedes Programm wird zunächst kurz beschrieben und später mit Hinweisen auf Änderungsmöglichkeiten der entsprechenden Programme aufgeführt.

77 BASIC-Programme, 208 Seiten, Softcover, DM 39,–
Der angegebene Preis ist der Ladenpreis.



te-wi Verlag GmbH
Telefon 089/1292090

te-wi

Theo-Prosel-Weg 1
8000 München 40

CP/M und WORDSTAR

Orientiert man sich im Computerbereich an Systemen amerikanischer Herkunft, die die Anwender-Zielgruppe unterer und teilweise mittlerer Bereich betrifft, stößt man immer wieder auf CP/M.
Für dieses Software-System zum Betrieb von Computern ist nun ein Standardwerk in deutscher Sprache erschienen, das dem ständig steigenden Kreis von Mikrocomputer-Benutzern eine fundamentale Einarbeitungshilfe bietet. Dieses Buch baut auf den ursprünglichen Unterlagen über CP/M von Digital Research einerseits auf, setzt diese jedoch nicht voraus. Es werden die grundsätzlichen Zusammenhänge des BIOS beschrieben, wie auch das Mehrbenutzer-Betriebssystem MP/M, das auf CP/M aufbaut. Als logische Vervollständigung rundet die Beschreibung über das Textsystem „WordStar" den gebotenen Inhalt sinnvoll ab.
Für alle, die das Riesen-Reservoire an durchweg kommerzieller Software beanspruchen und darüberhinaus selbst Programme entwickeln wollen.

CP/M und WORDSTAR – von Rüdiger Paul und Martin Riedel
144 Seiten, Paperback, DM 29,80 (der angegebene Preis ist der Ladenpreis)

---

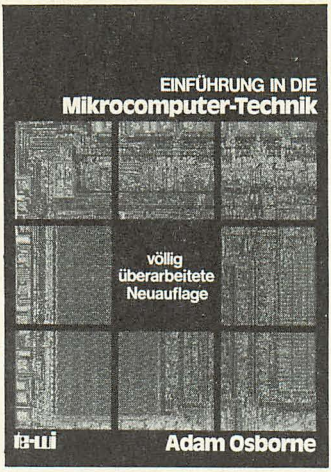
te-wi Verlag GmbH
Telefon 089/1292090

te-wi

Theo-Prosel-Weg 1
8000 München 40

EINFÜHRUNG IN DIE MIKROCOMPUTER-TECHNIK

VÖLLIG NEU ÜBERARBEITET!



Wie konnte dieses Buch Bestseller in den USA werden? Nicht nur die international anerkannte Kompetenz des Autors, Adam Osborne, auf dem Gebiet der Mikrocomputer-Technik gibt die Erklärung – sondern seine ungewöhnliche Fähigkeit, eine der bedeutsamsten Technologien unserer Zeit mehr in Bildern statt Worten erklären zu können. Beim Durchblättern dieses Buches werden Sie kaum einen Gedanken finden, der nicht als Bild anschaulich gemacht wurde. Diese Einführung in Prinzipien, Funktion und Systemaufbau von Mikrocomputern vermittelt Ihnen eine lebendige und umfassende Vorstellung vom Wesen dieser Technik und befähigt Sie, konkrete Aufgaben durch richtige Fragen an Experten oder durch Verständnis der Systembeschreibungen von Mikrocomputern zu lösen. Sie werden nicht nur die Komponenten eines Mikrocomputersystems richtig beurteilen und für Ihre Zwecke geeignet auswählen können, sondern auch zu Fragen der Programmierung, Vergleich alternativer Mikrocomputersysteme, Fragen der Wirtschaftlichkeit bei Anwendung in Massenprodukten und zur Beurteilung zukünftiger Entwicklungen mit Sachverstand Stellung nehmen können. Dieses bestens bekannte Standardwerk des Autors, das auch an über 500 Universitäten weltweit als reguläre Studiengrundlage eingeführt ist, hat eine absolute Überarbeitung erfahren. Es spiegelt den allerletzten Stand der faszinierenden Mikrocomputer-Technik wider.

EINFÜHRUNG IN DIE MIKROCOMPUTER-TECHNIK – GRUNDLAGENBUCH
völlig überarbeitete Neuauflage
von Adam Osborne
488 Seiten, zahlreiche Abbildungen, Hardcover, farbiger Schutzumschlag,
DM 66,– (der angegebene Preis ist der Ladenpreis)

---

te-wi Verlag GmbH
Telefon 089/1292090

te-wi

Theo-Prosel-Weg 1
8000 München 40

MIKROCOMPUTER–GRUNDWISSEN

Eine allgemeinverständliche Einführung in die Mikrocomputer-Technik – vom Mikrocomputer-Papst Dr. Adam Osborne.
Optimal als Einstieg für Elektronik-Laien; zur Kontaktaufnahme mit diesem die Technik und unsere Umwelt revolutionierenden Gebiet.
Dieses Buch ist aber kein trockener Lernstoff, sondern in seiner didaktisch ausgezeichneten und humorvollen Form – mit vielen Illustrationen und Cartoons – ein Werk, das man verschlingt. Und dabei spielend leicht ein Thema begreift, das unser Leben immer mehr beherrscht.

MIKROCOMPUTER-GRUNDWISSEN – von Adam Osborne
304 Seiten, Paperback, DM 36,– (der angegebene Preis ist der Ladenpreis)

---

te-wi Verlag GmbH
Telefon 089/1292090

te-wi

Theo-Prosel-Weg 1
8000 München 40

APPLE II PASCAL

Diese praktische Anleitung wendet sich an alle, die die Programmiersprache PASCAL lernen wollen. Für die praktischen Übungen muß ein Apple II Computer zur Verfügung stehen.
Der Leser lernt Schritt für Schritt – anhand von einfachen Programmen – die Grundlage und auch die Feinheiten der Sprache kennen, daneben wird er in das Apple-PASCAL-Betriebssystem eingeführt.

Nach Durcharbeitung der ersten Hälfte des Buches ist er in der Lage, selbständig Programme zu schreiben. Ein empfehlenswertes Buch, besonders für Anfänger, da es klar und einfach, teilweise sogar nicht ohne Humor, geschrieben ist.

APPLE II PASCAL – Eine praktische Anleitung
von Arthur Luehrmann – Herbert Peckham
544 Seiten, Paperback, DM 59,– (der angegebene Preis ist der Ladenpreis)

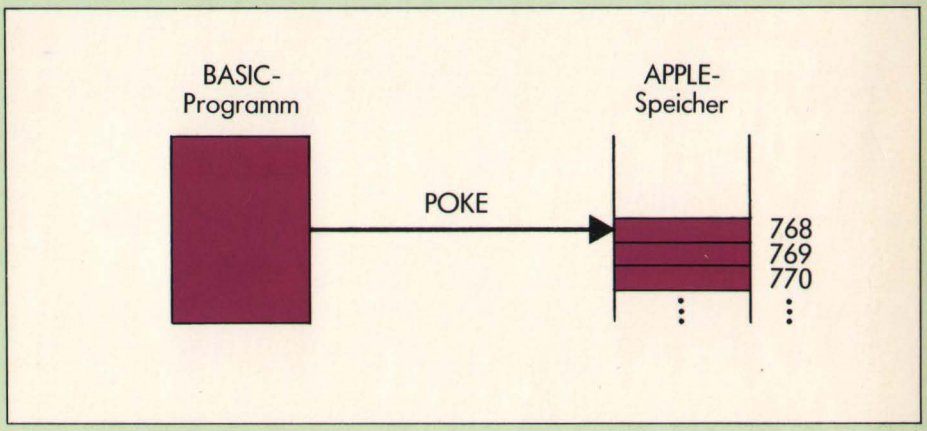
---

te-wi Verlag GmbH
Telefon 089/1292090

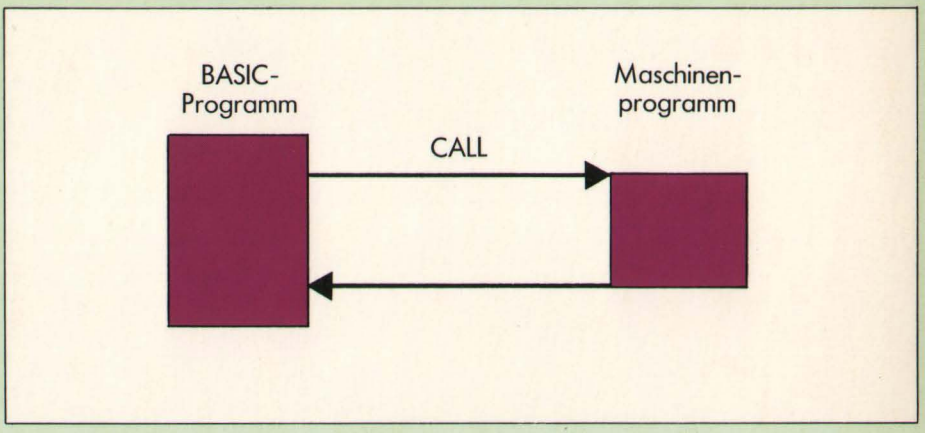
te-wi

Theo-Prosel-Weg 1
8000 München 40

Ein BASIC-Programm kann Maschinenbefehle mit dem Befehl **POKE** im APPLE-Computer abspeichern:



und dann mit dem BASIC-Befehl **CALL** in dieses Maschinenprogramm springen, als wäre es ein BASIC-Unterprogramm:



Dieses Buch bildet die einfachste Brücke zwischen dem vertrauten BASIC und der Maschinensprache des Mikroprozessors 6502 in Ihrem APPLE-Computer.

ISBN 3-921803-21-7